MACHINE LEARNING: TECHNIQUES AND FOUNDATIONS

Pat Langley
Irvine Computational Intelligence Project
Department of Information & Computer Science
University of California, Irvine, CA 92717

Jaime G. Carbonell
Department of Computer Science
Carnegie-Mellon University
Pittsburgh, Pennsylvania 15213

Technical Report 87–09

March 30, 1987

Abstract

The field of *machine learning* studies computational methods for acquiring new knowledge, new skills, and new ways to organize existing knowledge. In this paper we present some of the basic techniques and principles that underlie AI research on learning, including methods for learning from examples, learning in problem solving, learning by analogy, grammar acquisition, and machine discovery. In each case, we illustrate the techniques with paradigmatic examples.

An earlier version of this paper appeared in S. Shapiro (Ed.), *Encyclopedia of Artificial Intelligence*, John Wiley & Sons.

This work was supported in part by contract N00014-82-C-50767 and contract N00014-84-K-0345 from the Office of Naval Research, and in part by a gift from the Hughes Corporation.

We would like to thank Stephanie Sage and Nancy Wong for typesetting and proofreading this paper, and Kim Shin for drawing some of the figures.

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER Technical Report No. 10	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE *(and Subtitle)* Machine Learning: Techniques and Foundations		5. TYPE OF REPORT & PERIOD COVERED Final Report 1/84–1/87 6. PERFORMING ORG. REPORT NUMBER UCI–ICS Technical Report 87–09
7. AUTHOR*(s)* Pat Langley and Jaime G. Carbonell		8. CONTRACT OR GRANT NUMBER*(s)* N00014–84–K–0345
9. PERFORMING ORGANIZATION NAME AND ADDRESS Department of Information & Computer Science University of California, Irvine, CA 92717		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Information Sciences Division Office of Naval Research Arlington, Virginia 22217		12. REPORT DATE March 30, 1987 13. NUMBER OF PAGES 51
14. MONITORING AGENCY NAME & ADDRESS *(if different from Controlling Office)*		15. SECURITY CLASS. *(of this report)* Unclassified 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT *(of this Report)*

Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20, if different from Report)*

18. SUPPLEMENTARY NOTES

An earlier version of this paper appeared in S. Shapiro (Ed.), *Encyclopedia of Artificial Intelligence*, John Wiley & Sons.

19. KEY WORDS *(Continue on reverse side if necessary and identify by block number)*

learning from examples
heuristics learning
learning by analogy
grammar acquisition
machine discovery
credit assignment
analytic learning
conceptual clustering

20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)*

The field of *machine learning* studies computational methods for acquiring new knowledge, new skills, and new ways to organize existing knowledge. In this paper we present some of the basic techniques and principles that underlie AI research on learning, including methods for learning from examples, learning in problem solving, learning by analogy, grammar acquisition, and machine discovery. In each case, we illustrate the techniques with paradigmatic examples.

1. Reasons for Studying Machine Learning

One of the defining features of intelligence is the ability to learn. Thus, machine learning is of central concern to the field of artificial intelligence. Upon closer inspection, three clear reasons for this concern become apparent. The first of these revolves around expert systems which, despite their success, often require man-years to construct and perfect. The bulk of the work goes into developing and debugging extensive domain-specific knowledge bases. A better understanding of the learning process might let us automate the construction of expert systems, and this in turn would greatly speed the development of applied AI systems.

The second reason for studying machine learning is more theoretical. Many AI researchers find expert systems unattractive because they lack the *generality* that science requires of its theories and explanations. On this dimension, the study of learning may reveal general principles that apply across many different domains. Artificial intelligence already acknowledges fairly general principles for problem solving and search, and machine learning holds the potential for similar principles.

A third research goal involves modeling human learning mechanisms. Generality is a central concern in this endeavor as well, since we know that all humans share a basic cognitive architecture but behave quite differently in similar circumstances. And a major determinant of such variation is the experience and level of knowledge of each individual. Thus, understanding human learning mechanisms provides one path towards explaining the invariant features of the human information processing system. However, useful applications would also emerge from a deeper understanding of human learning, since this would provide insights into the educational process, leading to better design of both classical teaching materials and intelligent automated tutoring systems.

2. A Brief History of Machine Learning

Before turning to recent work in machine learning, let us set the stage by reviewing the history of the field. The interest in computational approaches to learning dates back to the mid-1950's and the beginnings of artificial intelligence. However, early learning techniques tended to focus on numerical encodings and parameter tuning techniques. This contrasted with AI's growing emphasis on symbolic representations and heuristic methods, and in fact the early research on machine learning was more closely affiliated with the field of pattern recognition than it was with AI itself. Learning researchers were especially concerned with issues of generality, and attempted to construct systems that learned with very little initial knowledge.

This stage continued until the mid-1960's, when AI researchers first began to shift their attention to purely symbolic systems and knowledge-intensive approaches. In this period, most researchers avoided issues of learning while they attempted to understand the role of knowledge in constraining search. However, some work in machine learning continued in the

background, this time borrowing the symbolic representations and heuristic methods that had become central to artificial intelligence. It was during this stage that the first significant work on concept learning and language acquisition was carried out, and laid the basis for later efforts.

In the late 1970's, a new interest in machine learning emerged within AI and rapidly grew over the course of a few years. Research in concept learning and language acquisition continued, but this was joined by work on learning in the context of problem solving, as well as work on taxonomy formation, analogical reasoning, and machine discovery. Well-established methods were used to aid the construction of expert systems, and new methods were constantly formulated and tested. A substantial fraction of learning researchers had always been concerned with human learning, and this undercurrent continued into the new period. The number of published papers on machine learning increased dramatically, and the trend continues unabated.

With this brief history as context, let us now consider the problems and methods of machine learning in more detail. We have organized the paper according to five categorical tasks that have been addressed in the machine learning literature – learning from examples, learning search heuristics, learning by analogy, grammar acquisition, and learning by discovery. Taken together, these problem classes cover the vast majority of research that has been carried out in machine learning.[1]

In each case, we describe the learning task, consider the main methods that have been employed, and identify some open problems in the area. Although we have tried to convey the essence of these problems and methods, we encourage the reader to peruse other reviews of machine learning. These include papers by Mitchell (1982), Dietterich and Michalski (1983), and Carbonell, Michalski, and Mitchell (1983). We also direct the reader to two collected volumes of machine learning research (Michalski, Carbonell, & Mitchell, 1983, 1986) that present recent results in this active area.

3. Learning Concepts From Examples

The task of learning concepts from examples is the most widely studied problem in machine learning. Concept acquisition appears straightforward: given examples and counterexamples of some concept, generate an intensional definition of that concept. This definition should cover all the examples but none of the counterexamples, and it should correctly classify future instances. Despite its apparent simplicity, there are hidden complexities and multiple approaches; we consider the primary ones below. However, let us begin with an example to clarify the task and point out some of the problems.

---

[1] Unfortunately, we do not have the space required to describe all approaches to learning, and have focused on the most widely applied symbolic methods. In particular, we have omitted the work on genetic learning algorithms (Holland, 1975), connectionist models of learning (Hinton, Sejnowski, & Ackley, 1984), chunking and macro-operators (Fikes, Hart, & Nilsson, 1972; Laird, Rosenbloom, & Newell, 1986), learning from instruction (Mostow, 1983), and knowledge-acquisition aids for expert systems (Kahn, 1986).

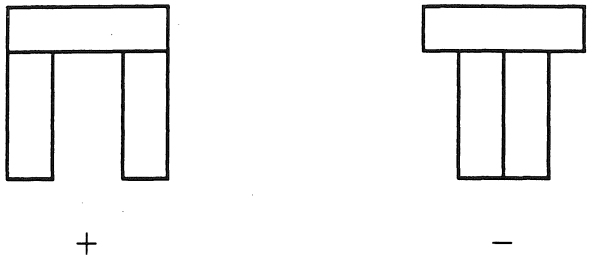


Figure 1. Positive and negative instances of "arch".

3.1 An Example: Learning the ARCH Concept

Perhaps the best known research on learning from examples is Winston's (1975) work on the "arch" concept. Figure 1 presents one example (*positive* instance) of this concept and one counterexample (*negative* instance). Given these instances, one might conclude that:

An ARCH consists of two non-touching vertical blocks and one horizontal block.

This intensional definition covers the positive instance and excludes negative instance. Of course, one *could* define ARCH extensionally, as the union of all positive examples of ARCH ever encountered. However, we would like our concept to be as simple as possible, and we would also like it to predict the classes of new instances. Although the initial definition given above is almost certainly incorrect, there is hope that it will eventually converge on the correct description of the concept.

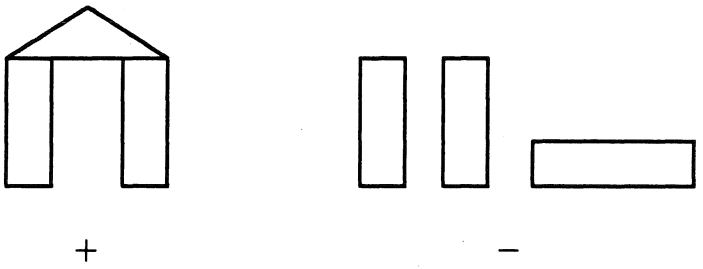


Figure 2. Additional positive and negative examples of "arch".

Now let us consider the two instances shown in Figure 2. Upon considering the positive instance, we realize that our concept of arch is too restrictive, since it excludes this instance. Therefore, we revise the concept to

An ARCH consists of two non-touching vertical blocks and one horizontal *object.*

However, this new hypothesis also covers the new negative instance, suggesting that it is overly general in some other respect. Revising the definition to exclude this nonexample, we might get:

An ARCH consists of two non-touching vertical blocks and a horizontal object *that rests atop both blocks.*

One can continue along these lines, gradually refining the concept to include all positive instances but none of the negatives. New positive instances that are not covered by the current hypothesis (errors of omission) tell us that the concept being formulated is overly specific, while new negative examples that are covered by the hypothesis (errors of commission) tell us it is overly general. We have not been very specific about how the learner responds to these two situations, but we consider some of the alternatives below. Most systems that learn from examples employ these two types of information, though we will see that they use them in quite different ways.

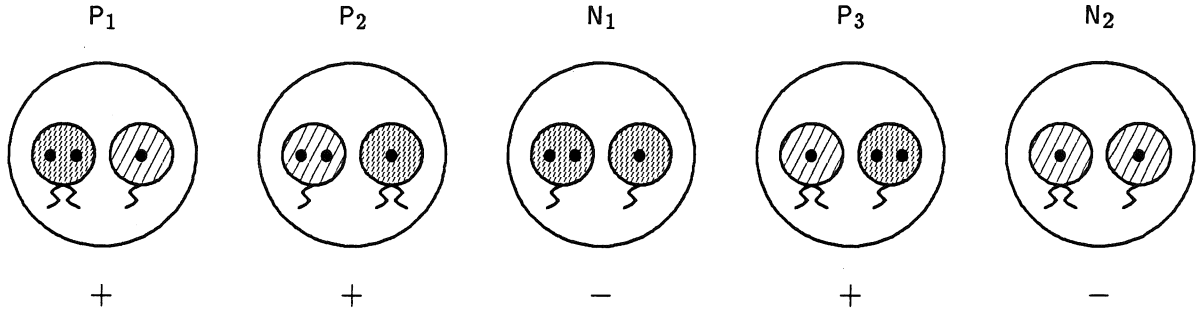


Figure 3. Positive and negative instances of diseased cells.

3.2 Learning from Examples as Search

As Mitchell (1982) and Dietterich and Michalski (1983) have pointed out, all AI systems that learn from examples can be viewed as carrying out *search* through a space of possible concepts. However, such 'hypothesis spaces' are unusual along a number of dimensions, and they are worth considering in more detail. For this purpose, we will use another example – learning to distinguish between diseased cells and healthy ones.[2]

Figure 3 presents five sample cells, three of which are identified with a disease (P_1, P_2, P_3), and two of which are not (N_1, N_2). Note that each cell contains two bodies, and that each of these bodies has three attributes – number of nuclei (one or two), number of tails (one or two), and its color (light or dark). The fact that each cell contains two bodies will prove important in our later discussions. Although we have used a graphical representation in the figure, one can also represent the instances in propositional terms. For example, we might represent P_1 as {(two two dark) (one one light)}, where the first term in each list stands for the number of nuclei, the second represents the number of tails, and the final term stands for the color. We have enclosed the two propositions in curly brackets to indicate that order is unimportant. Thus, the negative instance N_1 could be represented as either {(two one dark) (one one dark)} or as {(one one dark) (two one dark)}.

[2] This example is isomorphic to one presented by Mitchell (1982); we have simply replaced his features with biological ones.

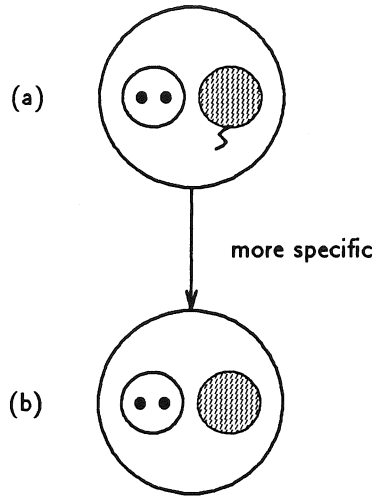


Figure 4. The generality ordering on concept descriptions.

Now that we have examined these instances, we can turn to the representation of concept descriptions (or hypotheses). Figure 4 presents two possible descriptions in the same graphical format as the instances. Note that only some of the features are present; this indicates that the missing features are considered irrelevant. Thus, hypothesis (a) states that for a cell to predict the disease, it must have one body with two nuclei and another body with one tail and a light color. In propositional terms, we can represent this description by {(two ? ?) (? one light)}, where ? means the value occupying that position is irrelevant. Now examine the bottom description (b). This has one fewer feature than hypothesis (a) and can be represented as {(two ? ?) (? ? light)}. As a result, the intensional definition (b) will cover more instances than (a). In such cases, we will say that (b) is *more general* than (a), and that (a) is *more specific* than (b).

Let us now consider the overall structure of this space of hypotheses. Figure 5 shows a number of states in the description space, with the most specific hypotheses at the top and the most general one at the bottom. Note that the most specific descriptions correspond to instances, since they have all features specified. In contrast, the most general hypothesis has no features given, and can be represented as {(? ? ?) (? ? ?)}. The most important thing to notice about this space is that the generality ordering is only *partial*. That is, although some hypotheses are related along the generality/specificity dimension, others are unrelated. For instance, hypotheses (a) and (b) in Figure 5 are both more specific than hypothesis (c), but neither is more specific than the other. It is this partial ordering that requires search through a sizable lattice of potential concept descriptions.

The generality dimension also suggests two classes of operators for moving through this problem space – one can make an existing hypothesis more general or one can make it more specific. These options also suggest two basic schemes for searching the space of concept descriptions. In the first, one begins with the most general hypothesis, and as new instances are encountered, more specific descriptions are produced. In the other, one begins with a very specific hypothesis, moving to more general descriptions as new data are observed. Both approaches take advantage of the partial ordering on hypotheses to constrain search,

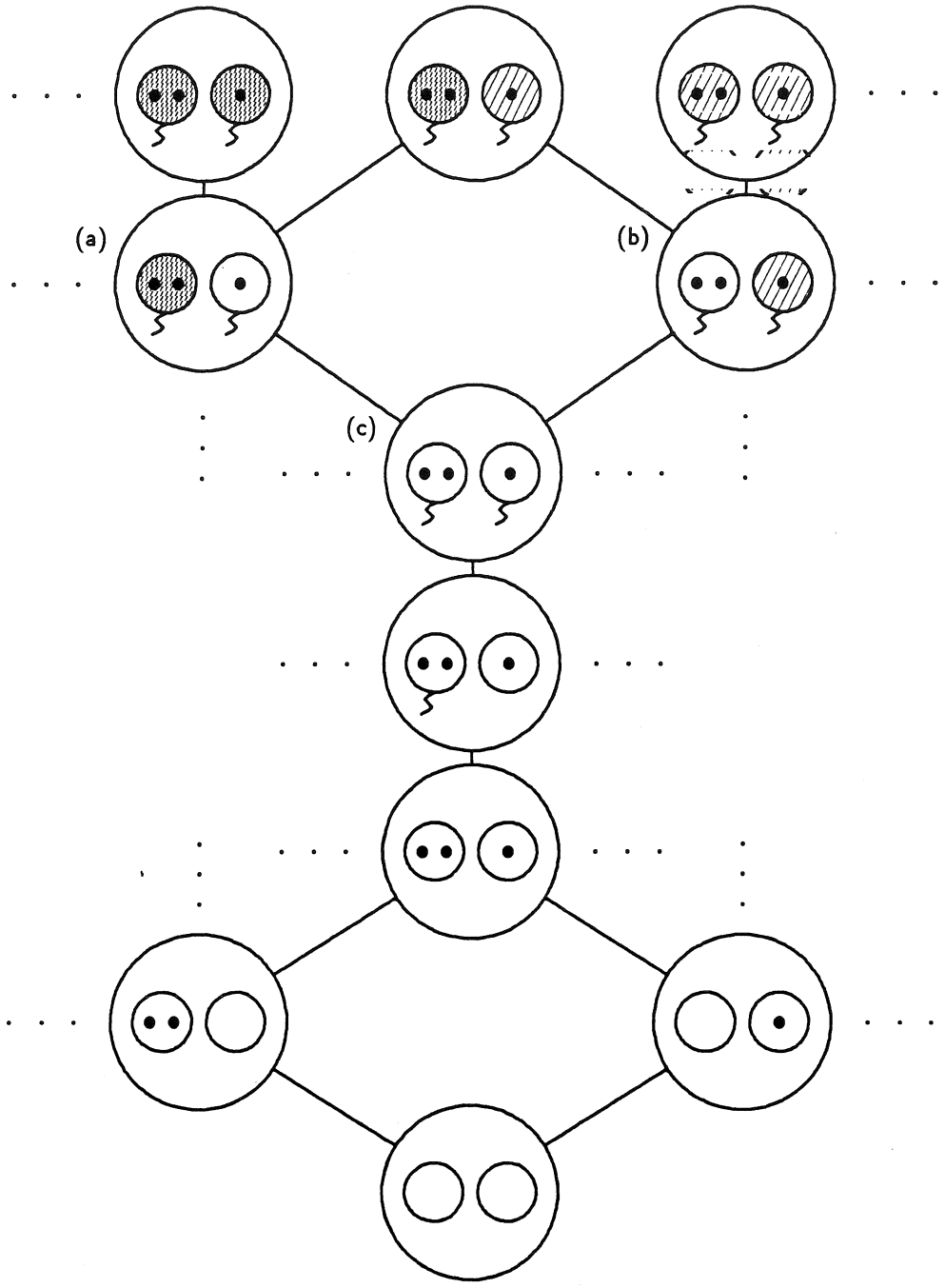


Figure 5. Partial ordering of the hypothesis space.

and most concept learning research has employed one or the other of these methods. Of course, one can combine both search directions, moving towards more general hypotheses in some cases and towards more specific ones at other times. Although most concept learning systems have organized search in the manner outlined above, a notable exception is the genetic algorithm approach (Holland, 1975), which does not use the partial ordering.

Taken together, the representation for states and the set of operators define a problem space (Newell & Simon, 1972), and we have examined the structure of this space for the task of learning from examples. However, one must still search the resulting space in an effective manner, and again a number of possibilities emerge. Some researchers have carried out a depth-first search through the concept space, while others (Mitchell, 1982) have used breadth-first search. Such exhaustive search methods are guaranteed to find the optimal concept definition, but may prove prohibitively expensive. Other researchers have used numeric evaluation functions to direct a heuristic search; for example, Michalski (1983) has used a beam-search method.

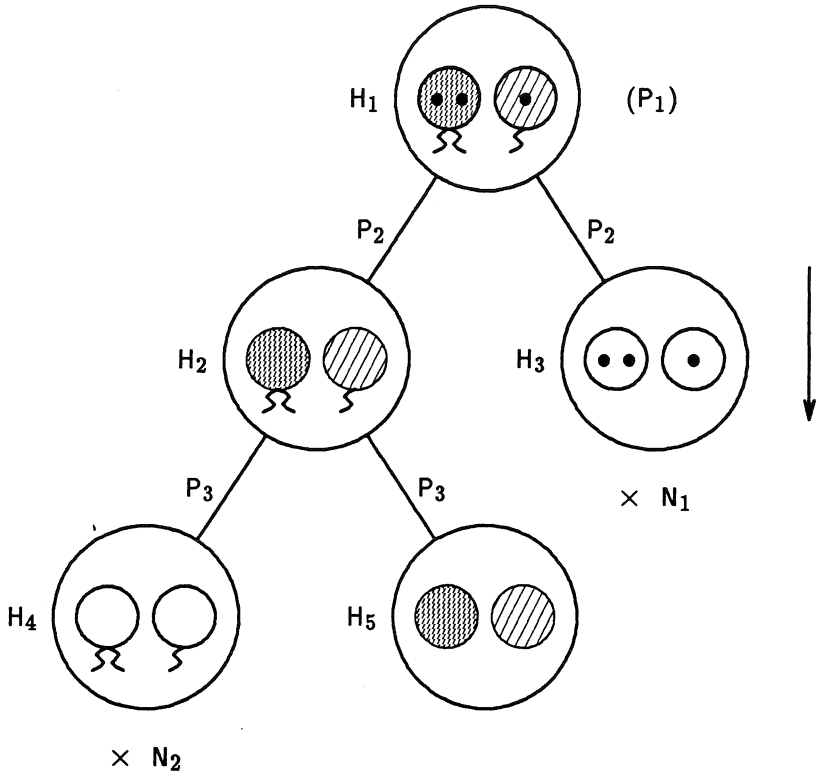


Figure 6. Searching from specific to general hypotheses.

3.3 Specific-to-General Methods

As we have seen, one can search the space of concept descriptions in two alternative directions – from specific descriptions to more general ones, or from general hypotheses to specific ones. Although there are many different ways to instantiate these basic methods, we will work with breadth-first versions, since they are the simplest for tutorial purposes. Let us begin the specific-to-general method. We will assume that the learner is presented with the instances from Figure 3 in an incremental fashion, and we will examine the active hypotheses after each instance has been processed.

In the specific-to-general scheme, the initial hypothesis is initialized to the first positive instance encountered by the learner. Thus, after observing instance P_1 from Figure 3, our

system would create the initial hypothesis H_1 shown in Figure 6. This description is very specific, and covers only the instance on which it was based. When the next positive instance (P_2) is encountered, the system notes that H_1 fails to match, suggesting that this hypothesis is overly specific. Accordingly, the learner removes H_1 and replaces it with more general hypotheses that cover both P_1 and P_2. Note that in this case, there are two ways [3] to make H_1 general enough to cover both instances, the first (H_2) ignoring the number of nuclei and the second (H_3) ignoring both the color and the number of tails. Also note that the new hypotheses are no more general than they need to be to cover the instances. That is, the minimally general hypotheses required to account for all the positive data are preferred over more general hypotheses that may later prove unwarranted. Most AI learning systems incorporate this principle of conservatism when generating hypotheses.

TABLE 1
Searching from Specific to General Hypotheses

Let H be the current set of hypotheses. Initialize H to the first positive instance p, and initialize the set of observed negative instances N to the empty set.

If p is the next positive instance, then:

1. For each hypothesis h∈H that does not match p, replace h with the most *specific* generalization(s) of h that will match p.
2. Remove from consideration all hypotheses that are more general than some other hypothesis in H.
3. Remove from H all hypotheses that match a previously observed negative instance n∈N, since they are overly general.

Step 1 is often accomplished by finding all *mappings* between p and h.

If n is a new negative instance, then:

1. Add n to the set of negative instances N.
2. Remove from H all hypotheses that match n, since they are overly general.

While positive instances lead this algorithm to formulate more general hypotheses, negative instances let it eliminate competitors. For example, the fact that H_3 covers the nonexample N_2 suggests that this hypothesis is overly general in at least one respect. Since we are only allowing movement towards more general descriptions, the only option is to remove H_3.

[3] For specific-to-general methods this only occurs when two or more objects are involved, since it opens the possibility for multiple mappings between objects. E.g., one can map the left object in H_1 onto the left object in P_2 and the right object in H_1 onto the right object in P_2. However, one can also map the left object in H_1 onto the right object in P_2 and the right object in H_1 onto the left object in P_2. Each such mapping can lead to a different hypothesis. In contrast, if only a single object is involved, only one such mapping will be possible, eliminating the need for search.

Since H_2 does not cover N_2, it is retained for further expansion. Note that this strategy relies heavily on the assumption that all instances are correctly labeled. Most machine learning methods (but not all) rely on accurate data, making it difficult for them to handle noise.

After processing the first three instances, our learning system next encounters P_3. Since the only remaining hypothesis (H_2) fails to cover this positive instance, the system generates more general descriptions. As before, two new hypotheses are created, the first (H_4) referring only to the tails of the cell bodies and the second (H_5) referring only to their colors. When a second negative instance (N_2) is observed, the system notes that H_4 incorrectly covers the cell. As a result, it removes this hypothesis from consideration but retains H_5, since the latter correctly fails to cover the instance. One must also check to ensure that new hypotheses do not cover earlier negative instances, but H_5 fares well on this count as well.

The learner would continue in this mode, producing more general descriptions when positive instances require this action and eliminating hypotheses when negative instances are incorrectly matched. No learning occurs when correct predictions are made, since the existing hypotheses are performing as desired. Note that the system has no explicit means for deciding when it has acquired the final concept definition, but at each point it will have in memory the most specific hypotheses that account for all the data. Table 1 summarizes this specific-to-general method for learning from examples.

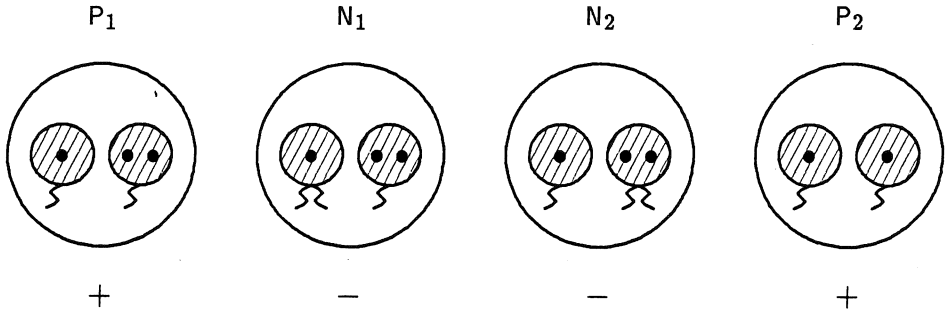


Figure 7. Additional positive and negative instances of diseased cells.

3.4 General-to-Specific Methods

Although the general-to-specific method differs from its alternative in the direction of search, the basic structures of the two methods are quite similar. As with the specific-to-general method, new instances sometimes lead to new hypotheses and sometimes lead to the elimination of existing descriptions. However, the roles played by positive and negative instances are reversed in the general-to-specific approach, as we will see below. For our example, we will use the set of instances shown graphically in Figure 7 and trace the development of the search tree presented in Figure 8.

In this approach, one begins with the most general hypothesis possible. In our sample domain, this is simply a cell with two bodies and no additional features specified, as shown in hypothesis H_1 at the bottom of Figure 8. After this initialization, the system processes

the first object in Figure 7, which happens to be the positive instance P_1. Since H_1 correctly covers this instance, no learning takes place at this point. However, it is useful to begin with a positive instance, since this will constrain search in successive steps.

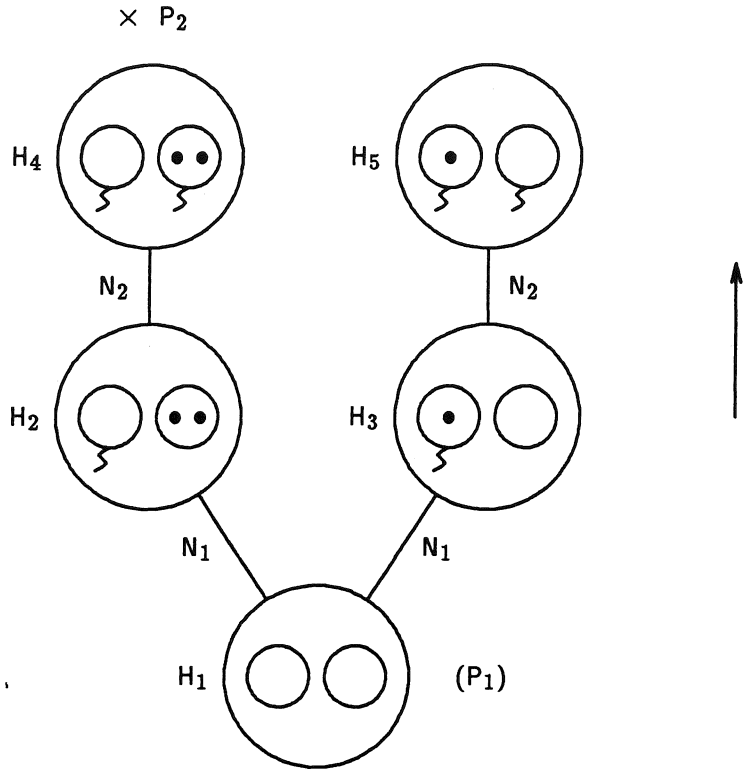


Figure 8. Searching from specific to general hypotheses.

The next instance (N_1) is negative. Our initial hypothesis is general enough to match this object (indeed, H_1 will match any cell), so we need to make it more specific. There are two ways to accomplish this and still cover positive instance P_1.[4] The first involves adding the features of "one tail" and "two nuclei" to the different cell bodies; the other involves adding "one tail" and "one nucleus" features to the same cell body. These new hypotheses are labeled as H_2 and H_3 in the figure. Note that in both cases we were required to add two features in order to rule out matches against N_1. Also note that these hypotheses are no more specific than necessary for this purpose; this is the principle of conservatism in operation again.

[4] In some cases, there will exist only one way in which to generate a more specific hypothesis. Winston (1975) has used the term "near misses" to describe negative instances that lead to this situation, and he has emphasized their role in reducing the search for concept descriptions. However, one cannot usually rely on their occurrence, so most learning systems have the ability to learn from "far misses" as well, albeit converging on the ultimate concept description more slowly.

Suppose our system next encounters the negative instance N_2 shown in Figure 7. Since both H_2 and H_3 incorrectly cover this instance, both must be made more specific. In both cases, there is only one way to accomplish this, each involving the addition of a single feature. The resulting concept descriptions are shown as H_4 and H_5 in Figure 8. Note that both of these cover the first positive instance P_1, which has been retained in memory as a constraint. Finally, the system encounters the new positive instance P_2, which matches against H_5 but not against H_4. This indicates that the first hypothesis is overly specific, and since we can only move towards more specific descriptions, it is removed from consideration.

This leaves H_5 as the only hypothesis, though the system would continue to make revisions as new data were gathered. As before, the method has no means to know when it has finished learning. Table 2 summarizes this general-to-specific method for learning from examples. Observe that this approach will generate the most *general* possible descriptions that cover the data, while the specific to general method will produce the most *specific* possible descriptions. Thus, one may want to employ one method or the other, depending on whether one desires optimistic or pessimistic rules. However, given a sufficiently rich sampling of the instance space, both methods eventually converge on the same concept description.

TABLE 2
Searching from General to Specific Hypotheses

Let H be the current set of hypotheses. Initialize H to the most general possible hypothesis, and initialize the set of observed positive instances P to the empty set.

If n is a negative instance, then :

1. For each hypothesis h∈H that matches n, replace h with the most *general* specialization(s) of h that will *not* match n.
2. Remove from consideration all hypotheses that are more specific than some other hypothesis in H.
3. Remove from H all hypotheses that fail to match a previously observed positive instance p∈P, since they are overly specific.

Step 1 is often accomplished by finding all *differences* between n and some p that is associated with h.

If p is a positive instance, then:

1. Add p to the set of positive instances P.
2. Remove from H all hypotheses that fail to match p, since they are overly specific.

3.5 Combining the Approaches

As we mentioned earlier, one can combine the specific-to-general and the general-to-specific approaches for searching the space of concept descriptions. The combination provides some advantages that neither method exhibits in isolation. For example, Anderson and

Klein (1979) have employed a combined approach in which one begins with specific rules or hypotheses, generates more general descriptions as new positive instances are observed, and then produces more specific hypotheses when these prove to be overly general. This scheme simulates a form of backtracking without the need for memory of the search path, and they have also used it to generate disjunctive descriptions.

Mitchell's (1978) version-space method combines the two techniques in a quite different manner. This method retains two sets of hypotheses – the most specific set of descriptions that cover the data (S) and the most general set of such descriptions (G). When new positive instances are encountered that are not covered by elements of S, the method uses the first algorithm described above to transform those elements into more general descriptions. Similarly, when new negative instances covered by some element in G come into play, the second algorithm above leads to more specific versions of the G set. One notable difference is that one no longer need retain either positive or negative instances. The S set summarizes the positive data and is used to eliminate overly specific members of the G set, while the G set summarizes negative instances and is used to detect overly general members of the S set.

The version-space approach has two interesting features. First, one knows when the learning task has been completed; this occurs when the S and G sets converge on a single concept description. Second, although the members of S and G have identical forms to the hypotheses we have been discussing, their interpretation is somewhat different. Rather than representing hypotheses about the concept itself, members of the S and G sets act as *boundaries* on the space of descriptions that are consistent with the data. As more instances are gathered, these boundaries become more constraining, until eventually they eliminate all but one concept description. Basically, the version-space method employs a *constraint satisfaction* approach to learning from examples, in contrast to the simpler search-based methods we described above.

3.6 Constructing Decision Trees

Most research on learning from examples has employed some variant on the approaches described above. However, it would not be fair to leave this topic without mentioning another quite different class of methods. Quinlan (1986) has called these TDIDT algorithms, which stands for *Top-Down Induction of Decision Trees.* As the name suggests, these learning methods represent concepts not as conjunctions of conditions, but rather as decision trees. In addition, these systems construct their trees in a top-down fashion, and they are nonincremental in that they require all instances to be present at the outset.

The earliest work in the TDIDT tradition was carried out by Hunt, Marin, and Stone (1966), but Quinlan's ID3 (1983, 1986), is the best known of these systems, so we will focus on it here. The input to ID3 is a list of positive and negative instances of some concept, with each instance represented as a list of attribute-value pairs like those shown in Table 3. The output is a decision tree like the one in Figure 9, with tests at each node for sorting instances down alternative branches. Terminal nodes specify the class of objects that have been sorted to that location.

TABLE 3
Sample data for ID3 [After Quinlan (1986)]

outlook	temperature	humidity	windy	class
sunny	hot	high	false	−
sunny	hot	high	true	−
overcast	hot	high	false	+
rain	mild	high	false	+
rain	cool	normal	false	+
rain	cool	normal	true	−
overcast	cool	normal	true	+
sunny	mild	high	false	−
sunny	cool	normal	false	+
rain	mild	normal	false	+
sunny	mild	normal	true	+
overcast	mild	high	true	+
overcast	hot	normal	false	+
rain	mild	high	true	−

Quinlan's algorithm begins with only the top node of a network, and grows its decision tree in a top-down manner, one branch at a time. At each point, it uses an information-theoretic evaluation function to determine the most discriminating attribute; this score is based on the numbers of positive and negative instances associated with the values of each attribute. For example, the instances in Table 3 contain four attributes – *outlook*, *temperature*, *humidity*, and presence of *wind*. Upon inspecting these instances, the method decides that the *outlook* attribute does the best job of distinguishing between the positives and negatives. As a result, it creates three branches at the top level of its decision tree, one for each value of the *outlook* attribute.

The instances are then sorted down the appropriate branches, and the algorithm checks to see whether all instances at a given node are positive. If so, this node is marked as terminal and labeled as leading only to positive instances. An analogous step is taken when all instances sent to a node are negative. In Figure 9, this occurs with the *overcast* value of the *outlook* attribute, which contains only examples of the concept. However, if both positive and negative instances are shipped to a node, the tree-building process is applied recursively to this subset of the data. This occurs with both the *sunny* and *rain* values in our example. For the *sunny* subset, the next most discriminating value is the *humidity* attribute, while the *windy* attribute has the best score for the *rain* subset. In both cases, the resulting sets contain only one type of instance, causing all nodes to be labeled as terminal and thus halting the tree-building process.

The TDIDT approach differs from the methods we described earlier along a number of dimensions. First, decision trees can easily represent many forms of disjunction, and systems like ID3 have no trouble acquiring such concepts, despite the difficulty experienced by other

methods we have examined. Second, the TDIDT scheme can be easily adapted to handle noise and Quinlan (1986) has run extensive experiments that show this capability. This results from the nonincremental nature of the algorithm, which lets one compute statistical measures. Another advantage is that ID3 and its relatives carry out very little search, relying instead on a numeric evaluation function to select the best attribute at each point. This is in marked contrast to the breadth-first searches carried out by Mitchell's (1978) version space method and similar techniques.

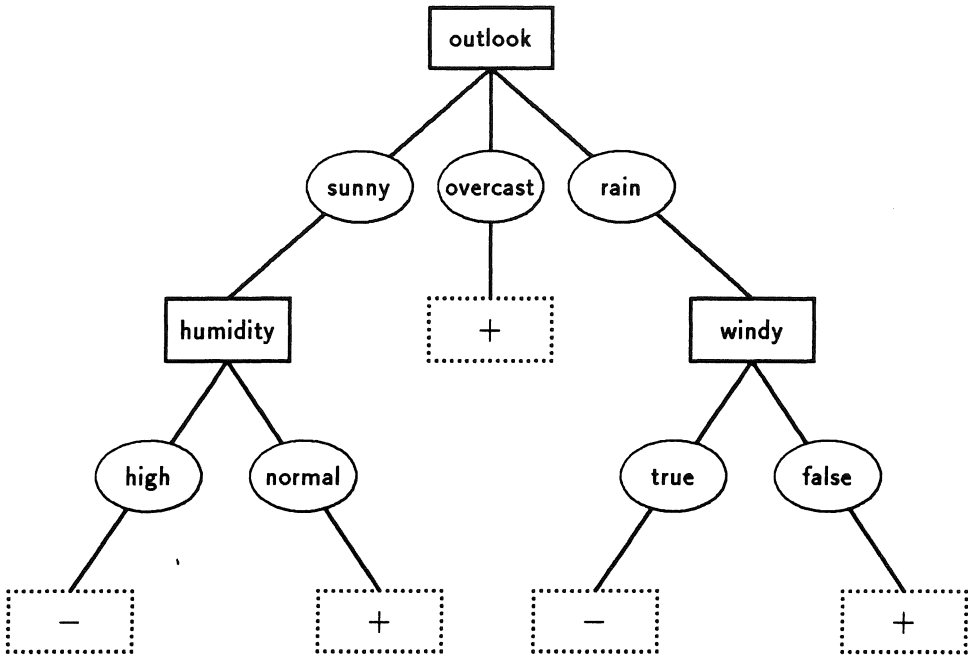


Figure 9. A sample ID3 decision tree.

As might be expected, the TDIDT approach suffers from a different set of limitations. For instance, it is restricted to attribute-value representations, while the other schemes we have examined can deal with relational descriptions. Also, the nonincremental nature of TDIDT systems make them quite inefficient at incorporating new instances, since they must recompute their trees from scratch when new data are encountered. In other words, tradeoffs exist between the two approaches, and neither is superior to the other in any absolute sense.

3.7 The A^q Algorithm

The TDIDT approach is not the only nonincremental method for learning from examples. Michalski and his colleagues have developed a family of programs based on the A^q algorithm, which they have tested on a variety of learning tasks (Michalski, 1975; Michalski & Larson, 1978), including some involving noisy data. Instead of producing a decision tree, the algorithm generates a concept description stated as a disjunction of conjunctions, which is equivalent to a set of production rules.

The A^q method starts by selecting some *seed* object from the positive instances and finding an optimal concept description that covers this instance but none of the negative instances. The algorithm then removes all positive instances that are covered by the description, selects a new seed from the remaining set, and generates another concept description based on the new seed. This process continues until all positive instances have been covered by at least one of the generated descriptions.

Michalski's algorithm carries out a beam search through the space of descriptions and, like Quinlan's method, it uses an evaluation function to constrain its search. The default preference criterion favors conjunctive descriptions that cover the maximum number of positive instances. Therefore, if the algorithm cannot find a single conjunctive concept description, it generates a description with a small number of disjunctions. The A^q method carries out considerably more search than ID3 and its relatives, but it can also handle relational descriptions, and Michalski and his collaborators claim that its concept descriptions are easier to understand than decision trees. They have also described incremental versions of the A^q algorithm (Michalski & Larson, 1978; Reinke & Michalski, 1986).

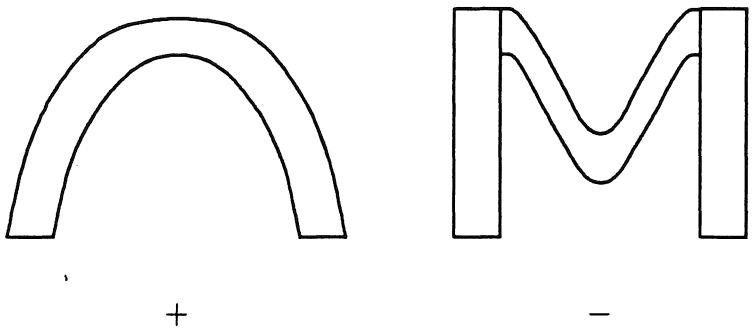


Figure 10. More instances of "arch" – the need for functional definitions.

3.8 Analytic Approaches to Concept Learning

Nearly all research on learning from examples has focused on *structural* definitions of concepts, but humans clearly also employ knowledge of different *functions* that objects and actions might assume in recognition and learning. In order to clarify this point, let us return briefly to our ARCH example. Figures 1 and 2 presented positive and negative instances of the ARCH concept, and we found it fairly easy to formulate a structural description of ARCH from these examples. But now consider the positive and negative instances in Figure 10. Structurally, neither object has much in common with the earlier ones, but in functional terms the object on the left makes a fine ARCH, while the rightmost object fails miserably. For a system to grasp such categorical distinctions, it must have knowledge about how arches are used and about why one might build them.

Recent research in machine learning has started to address this problem, examining ways to transform functional definitions into structural ones using a minimum of instances. This approach has been called *analytical* learning by some researchers, while it has been labeled *explanation-based* learning by others. These methods are analytical in the sense that they use

considerable domain knowledge to reason about *why* an object or event is a positive instance of a concept. They are explanation-based in the sense that they construct *explanations* of why an object satisfies a functional definition, and use this explanation to formulate a structural description.

TABLE 4
An Example of the Analytic Approach

Given:

- *Goal concept:* The concept of a 'cup.'
- *Training Example:*
  PART-OF(OBJECT-1, CONCAVITY-1)
  COLOR(OBJECT-1, RED)
  CONCAVITY(CONCAVITY-1)
  UPWARD-POINTING(CONCAVITY-1)
  OWNER(OBJECT-1, SAM)
  PART-OF(OBJECT-1, BOTTOM-1)
  BOTTOM(BOTTOM-1)
  FLAT(BOTTOM-1)
  LIGHT(OBJECT-1)
  PART-OF(OBJECT-1, HANDLE-1)
  HANDLE(HANDLE-1)
  LENGTH(HANDLE-1, 5)
- *Domain theory:*
  OPEN-VESSEL(O) & STABLE(O) & LIFTABLE(O) → CUP(O)
  PART-OF(O,B) & BOTTOM(B) & FLAT(B) → STABLE(O)
  PART-OF(O,C) & CONCAVITY(C) & UPWARD-POINTING(C) → OPEN-VESSEL(O)
  PART-OF(O,H) & HANDLE(H) & LIGHT(H) → LIFTABLE(O)
- *Operationality Criterion:* The concept must be defined in terms of predicates used in the example.

Find: An operational description of the goal concept that covers the training example.

The analytic approach requires a slight redefinition of the task of learning from examples. Although the output is still some intensional description of the concept, only a single positive is given as input. In place of the data used by empirical methods, the learner requires some *domain theory*, stated as a set of domain axioms and rules of inference that can be used to explain how an instance satisfies the concept. In addition, the learning system needs some test for determining when an *operational* definition has been formulated. Usually this involves restating an initial functional definition in terms of structural features given in the training example.

Table 4 presents an instantiation of this task in which the concept to be learned is CUP; we have borrowed this example from Mitchell, Keller, and Kedar-Cabelli (1986). Note that the positive instance contains a number of irrelevant features, such as Sam being the owner; these will not be retained in the final concept description. Also note that the domain theory contains at least one rule with CUP in the right-hand side, and that it contains rules which mention features from the training example in the left-hand side. Both are necessary in order to generate an operational definition of the concept.

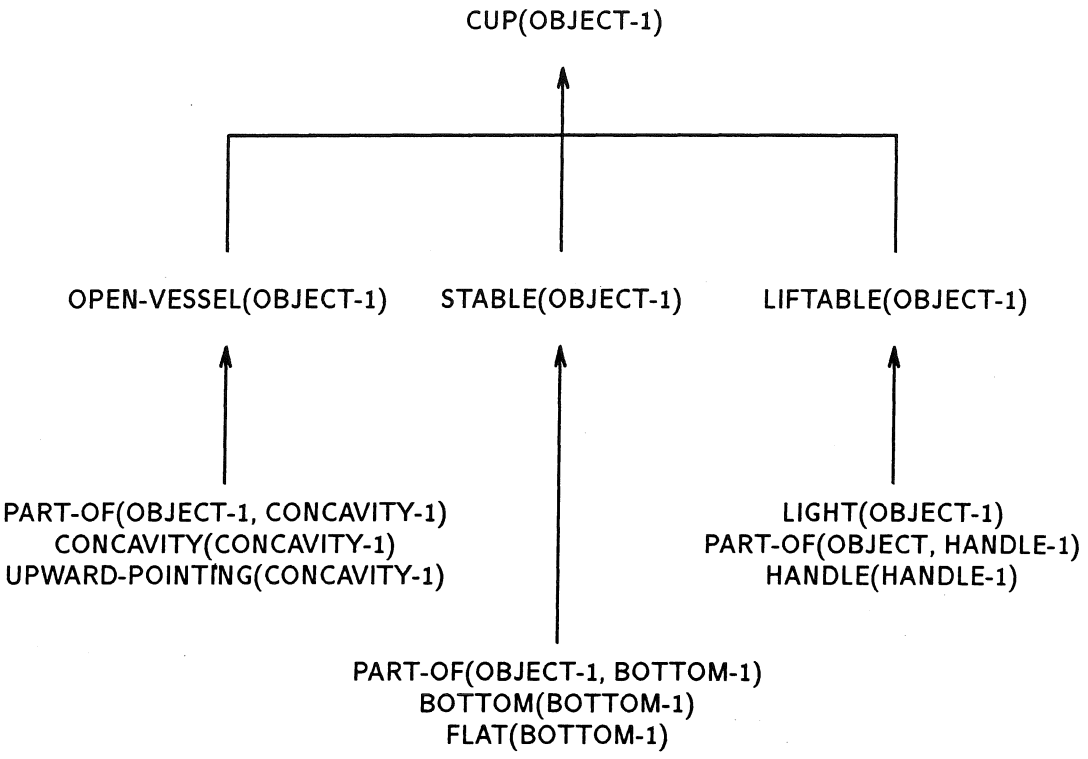


Figure 11. Explanation for an instance of "cup."

The basic approach involves a two-step process. First one uses the domain theory to construct an explanation[5] that proves the training example is a positive instance of the goal concept. The terminal nodes of the resulting explanation tree must be operational. Figure 11 presents such an explanation for the positive instance of CUP given in Table 4. Note that the top node in the explanation tree refers to CUP, while each of the terminal nodes refer to propositions in the training example. The second step involves transforming these terminal nodes into a set of sufficient conditions under which the explanation will hold, i.e., finding the most general version of the existing proof consistent with the domain theory. This can be accomplished by regressing the goal concept through the explanation tree, replacing constants with variables when appropriate and unifying variables that are required to match

[5] Such an explanation is actually a proof that the concept applies to the instance, given the axioms of the domain theory.

against the same term. This is an especially simple example, since the explanation is only two levels deep, but more complex cases can also be handled.

The analytic approach to learning from examples has a number of advantages over empirical methods. First, it provides a logical justification for the concept description that is formed, while empirical approaches make 'unjustified' inductive leaps from the data. Second, explanation-based methods can learn from a single positive instance; negative instances are not required at all. Analytic methods also handle disjunctive concepts, since they find only *sufficient* (rather than necessary) conditions on the concept. Although a new explanation will be required when another disjunct is encountered, there is no assumption that the concept can be described conjunctively. Finally, the approach handles noisy data, since the explanation process will be unable to explain misclassified instances and will thus discard them.

However, the analytic approach to learning from examples also has some disadvantages compared to empirical methods. For one, it requires significant domain knowledge and thus can be applied in fewer situations. Whereas analytic methods require little search through the space of concept descriptions, they require significant search through the space of explanations. Thus, clear tradeoffs exist between the two paradigms for learning. One direction for future research is the development of combined empirical/analytical methods, which may give the best of both worlds.

Before closing our discussion of learning from examples, we should draw attention to yet another difference between the two methods. Empirical learning methods move from specific data to some general rule or description, and in this sense they are clearly doing *induction*, whether they search the description space in a general-to-specific or a specific-to-general direction. In contrast, analytic learning methods transform some general description (e.g., a functional definition) into some other general description (e.g., an operational one). Most analytic techniques use a training example to focus their attention and limit the search for explanations, but this is not required in principle. Thus, analytic methods can be viewed as learning by *deduction* rather than by induction. This is an important distinction, but we do not yet understand its full implications.

3.9 Some Open Problems in Learning from Examples

A number of problems remain to be addressed with respect to learning from examples. Most of these relate to simplifying assumptions that have typically been made about the concept learning task. For instance, many researchers have assumed that no noise is present (i.e., all instances are correctly classified). However, there are many real-world situations in which no rule has perfect predictive power, and heuristic rules that are only usually correct must be employed. Some statistically-based learning methods (such as Quinlan's) can be adapted to deal with noisy data sets, whereas the incremental methods (such as the version space method) seem less adaptable. Schlimmer and Granger (1986) and Langley (1987) have described incremental methods that appear robust with respect to noise, but more remains to be done. In addition, tradeoffs appear to exist between the ability to deal with noise

and the number of instances required for learning, and it would be useful to know the exact nature of such relationships.

A related simplification is the assumption that the correct language for representing the concept is known in advance. If a learning system employs an incomplete or incorrect representation for its concepts, then it is searching a rule space that does not contain the desired concept. One approach is to construct as good a rule as possible with the representation given; a system that can deal with noise can handle incomplete representations in this manner. A more interesting approach is one in which the system gradually improves its representation language. This is equivalent to changing the space of rules one is searching, and on the surface at least, appears to be a much more challenging problem. Little work has been done in this area, but Utgoff (1983) and Lenat (1983) have made an interesting start on the problem.

Another simplifying assumption that nearly all concept learning researchers make is that the concept to be acquired is *all or none*. In other words, an instance either is an example of the concept or it is not; there is no middle ground. However, almost none of our everyday concepts are like this. Some birds fit our bird stereotype better than others, and some chairs are nearer to the prototypical chair than others. (Is a Dodo a bird? Is a Platypus a better bird? If a person sits on a log, is it a chair? Is it a better chair if we add stubby legs and use a second log as a backrest?) Unfortunately, nearly all existing concept learning systems rely on the sharp and unequivocal distinction between positive and negative instances, and it is not clear how they might be modified to deal with fuzzily-defined concepts such as birds and chairs. This is clearly a challenging direction for future research in machine learning, one where the functional approach to concept classification enjoys some advantages over the purely structural approach.

4. Learning and Problem Solving

Although the recognition and formation of concepts is a central component of intelligent behavior, there are other equally important aspects, such as the ability to solve problems and formulate plans. As humans gain experience in a domain, they improve their ability to solve problems in that domain, and we would like machine learning systems that also improve their problem solving skills. Most AI research on problem solving has focused on methods for *searching* some problem space, and within this framework there are three clear roles for learning. All three rely on the notion of an *operator* for moving through states towards a goal in a problem space.

The first approach involves learning *heuristic conditions* on operators in order to direct the search process. The second involves acquiring *macro-operators* in order to increase the size of the steps taken through the problem space. The third approach entails *analogical transfer* of expertise across similar problem domains. First we focus on the heuristics learning task, and in the next section we address learning by analogy.

TABLE 5
Learning Heuristics for Symbolic Integration

Given: A problem space for integration:

- Initial states:
  - $\int x^2 \, sin(x) \, dx$
  - $\int cos^3(x) \, dx$
- A set of operators:
  - $\int r \, f(x) \, dx \rightarrow r \, int f(x) \, dx$
  - $\int u \, dv \rightarrow u \, v - \int v \, du$
  - $f^r(x) \rightarrow f(x) \, f^{r-1}(x)$
  - $\int sin(x) \, dx \rightarrow -cos(x)$
- Test for goals: No integral sign in the expression
- A search strategy: Breadth-first search

Find: Heuristic conditions for each integration operator.

4.1 The Task of Heuristics Learning

In order to understand the nature of heuristics and how they may be learned, we must recall that search involves *states* and *operators*. A problem is defined in terms of an initial state and a goal, and operators are used to transform the initial state into one that satisfies the goal. Search arises when more than one operator (or more than one instantiation of an operator) can be applied to a given state, requiring consideration of different alternatives. Some constraints are given in terms of the *legal* conditions on each operator, but these constraints are seldom sufficient to reduce search to tractable proportions for interesting problems. In order to accomplish this, the learner must also acquire *heuristic* conditions on the operators.

Table 5 states the heuristics learning task for the domain of symbolic integration, which has been studied by Mitchell, Utgoff, and Banerji (1983) and by Porter and Kibler (1986). Given a problem space for integration, one must find heuristic conditions on each integration operator. Consider the sample problem

$$\int x^2 \, sin(x) \, dx$$

and its solution as shown in Figure 12. This problem involves two applications of integration by parts, an especially nasty operator that always has at least two instantiations. The optimal solution path for this problem proceeds down the page, while undesirable steps are shown to the side. An ideal set of heuristics would guide the problem solver down the optimal path, ignoring the side paths.

4.2 Assigning Credit and Blame

The heuristics learning task is simplified by the nature of problem solving operators. In general, the operators tend to be independent of each other, and this suggests the *problem reduction approach* to heuristics learning: (1) divide the task into a number of subproblems, one for each operator; (2) formulate the heuristic conditions that determine the circumstances when an operator is useful; and (3) recombine the rules into a complete heuristic search system. Nearly all work on heuristics learning has taken this basic approach. Note that this scheme transforms the heuristics learning task into a number of *learning from examples* tasks, one for each operator. Thus, the approach requires positive and negative instances for each operator, and these must be computed in some fashion, since typically there is no tutor to provide them to the system.

In fact, the subtask of generating positive and negative instances for each operator is closely related to the *credit assignment problem*, a classical problem in machine learning from the early days of Samuel (1963) and his work on learning in the checkers domain. The problem occurs in situations where the learner receives feedback only after it has taken a sequence of actions. In order to improve its performance, the learner must assign *credit* to desirable actions and *blame* to undesirable ones, but this may not be easy. For instance, if one loses a chess game, the final move is seldom responsible for checkmate; usually some other (much earlier) move or set of moves led to this state, but identifying which move(s) may be very difficult. In any case, actions deserving credit can be viewed as positive instances of an operator, while actions deserving blame can be viewed as negative instances.

Given this framework, learning from examples can be viewed as an idealized case of heuristics learning, in which a single operator is involved and for which the solution path is but one step long. No true search control is necessary for the performance component, since feedback occurs as soon as a single "move" has been made. Credit assignment is trivialized, since the responsible component is easily identified as the rule suggesting the "move." However, the general problem is a very significant one, and learning from examples can be viewed as an artificial domain designed for studying the condition-finding problem in isolation from other aspects of the learning process. Heuristics learning is considerably more difficult than learning from examples, since the learner must generate its own positive and negative instances, and since multiple concepts (one per operator) must be acquired concurrently.

Within the problem reduction approach to heuristics learning, three basic solutions to the credit assignment problem have been explored: learning from solution paths; learning while doing; and the learning apprentice approach. Let us consider each of these methods in turn.

The first approach relies on waiting until a complete solution path to some problem has been found. Moves along the solution path are desirable since they lead the system toward the goal, while moves branching off the solution path are undesirable since they lead away from the goal. Thus, the method of learning from solution paths involves two steps: (1) marking every move along the solution path as a positive instance of the responsible operator; and (2) marking every move leading directly off the solution path as a negative instance of the

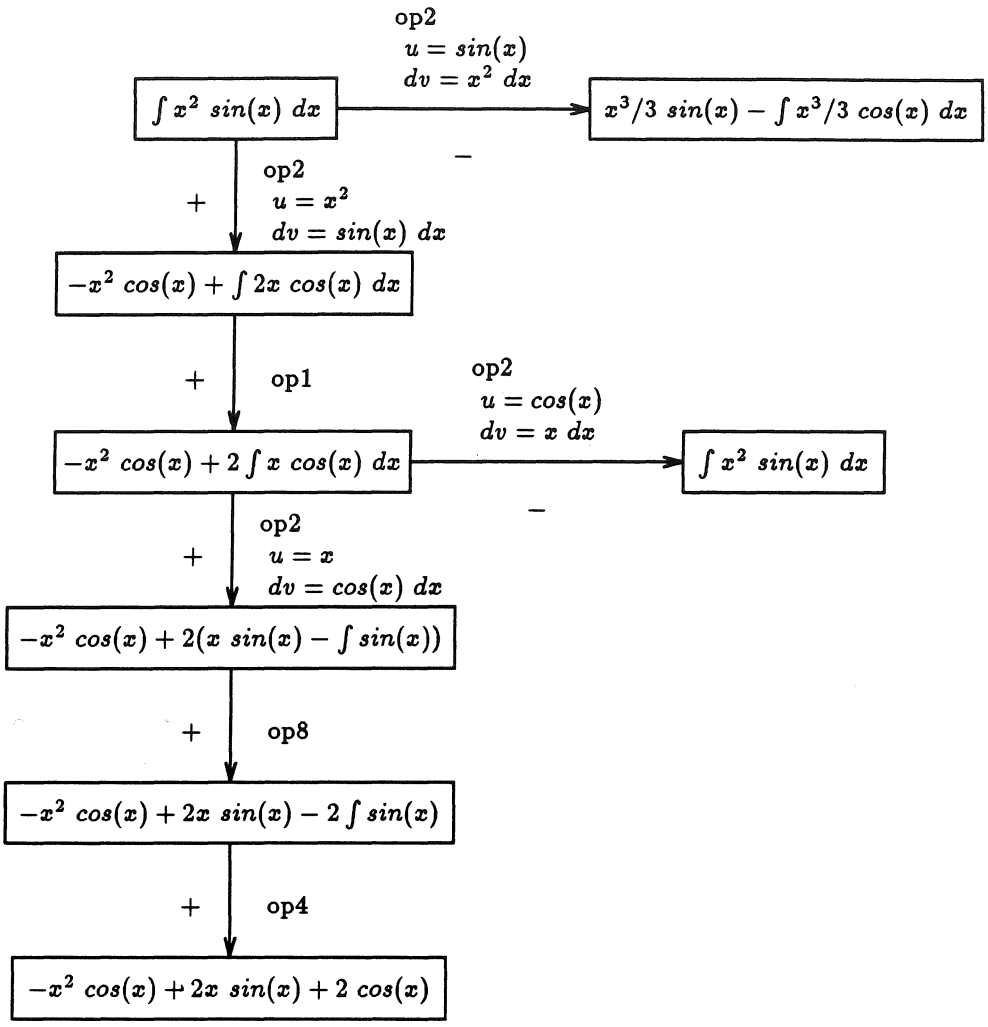


Figure 12. A solution path for an integration problem.

responsible operator. Note that this second decision is risky, since it is possible that a side path may lead to the goal by another route. For this reason, most systems that learn from complete solution paths rely on breadth-first search or some equivalent scheme to ensure that side paths do not also lead to the goal, or at least do not do so as efficiently. Also note that moves two steps off the solution path are ignored in this approach; the blame lies with the operator that led off the path in the first place, not with operators that were applied afterwards.

For example, consider the solution path for the integration problem shown in Figure 12. Moves along the solution path have been marked with a "+", while moves leading one step off this path have an associated "–". It is important to realize that the same operator may sometimes apply to a given state in multiple ways. Each of these is called an *instantiation*, and it is quite possible for one operator instantiation to be labeled as desirable and another to be marked as undesirable.[6] Thus, the integration by parts operator can be instantiated

[6] Instantiations depend upon the arguments to which an operator is applied. In chess, for instance, moving a rook requires stating which rook to move and what its destination square should be – one instantiation may be correct and another disastrous.

against the initial state in two different ways. One of these leads to the goal state, while the other leads away from it.[7] Other positive and negative instances of an operator may occur elsewhere along the path; in this problem, we see two other instances of the integration by parts operator, both applied to the third state along the solution path. Research in this tradition has been carried out by Anzai (1978), Mitchell, Utgoff, and Banerji (1983), and Langley (1985).

One limitation of the "learning from solution paths" approach is that it encounters difficulty in domains involving very long solution paths and extensive problem spaces. Obviously, one cannot afford to search exhaustively in a domain such as chess. In response, some researchers have explored methods that assign credit and blame while the search process is still under way. These techniques for *learning while doing* include schemes for noting loops and unnecessarily long paths, noting dead ends, and noting failure to progress towards the goal. For example, instantiating the operator for integration by parts can eventually lead one back to an earlier state, and this provides a clear opportunity for learning to avoid unproductive actions even before the goal has been achieved. Systems that incorporate such "learning while doing" methods include Anzai's (1978) HAPS, Ohlsson's (1983) UPL, Langley's (1983) SAGE.2, and Minton et al.'s (1987) PRODIGY. Ironically, with the exception of PRODIGY, these systems have all been tested in simple puzzle-solving domains, where the "learning from solution paths" method is perfectly adequate. Therefore, a promising research direction would involve applying these and other methods to more complex domains with long solutions and extensive search spaces.

The third credit assignment method involves observing an expert and using his actions to distinguish desirable moves from undesirable ones. Mitchell et al. (1983) have called this the *learning apprentice* approach, and it has natural applications to the semi-automated construction of expert systems. The advantage of this scheme is that it lets the learning system avoid excessive search, and at the same time provides immediate feedback about the desirability of moves. The disadvantage is that the learning system must rely on a tutor to lead it down the optimal solution path. In many ways, the learning apprentice approach transforms the heuristics learning task back into the simpler task of learning from examples, but this is sometimes useful. Other work in this paradigm has been carried out by Brazdil (1978), Neves (1978), Kibler and Porter (1983), and Minton et al. (1987).

4.3 From Instances to Heuristics

As we have seen, the process of assigning credit and blame to moves made during the search process is equivalent to labeling these moves as positive or negative instances of the responsible operators. Once these moves have been labeled, they can be used to determine heuristic conditions on each of the operators. The task is reduced to the problem of learning from examples. Thus, there will exist a space of 'concept descriptions' for each operator, in which the concept to be learned is 'those states under which the operator should be applied.'

[7] Actually, the negative instance also leads to the goal state, but by a more circuitous path. If one desires to learn heuristics for efficient integration, then it makes sense to label the instantiation leading down this inefficient path as a negative instance.

This space is partially ordered according to generality, and one can search the space using any of the methods we described earlier. Once the conditions for each operator have been identified, they can be used to direct search so the performance system prefers desirable moves to undesirable ones.

However, the task of heuristics learning does place some constraints on the method that is employed. In particular, the learning system must be able to generate both positive and negative instances of its operators. This poses no problem for general-to-specific systems, since they begin with overly general heuristics that lead naturally to search.[8] In contrast, specific-to-general methods are naturally conservative, preferring to make errors of omission rather than errors of commission. Such an approach works well if a tutor is present to provide positive and negative instances, but it encounters difficulties if a system must generate its own behavior. If a system relies on conservative heuristics to solve a problem, there will be many cases in which it refuses to make any move and problem solving will halt. Ohlsson (1983) has reported a mixed approach in which specific rules are preferred, but very general move-proposing rules are retained and used in cases where none of the specific rules are matched. However, in their pure form, specific-to-general methods do not seem appropriate for heuristics learning.

The majority of research on heuristics learning has focused on empirical methods, but there has also been recent work on analytic approaches to this problem. In this framework, one still requires a solution path from which to generate positive instances for each operator, but analytic methods are used to identify the heuristic conditions. Interestingly, one need not construct an explanation in these cases – the solution path itself suffices as the proof that a move was desirable, since it lies along the path to the goal state. Nor does one need additional domain rules, since the operators themselves play this role.

One must only reason backwards from the goal state, using the legal constraints on each operator to determine the features of each previous state that allowed the final operator in the sequence to apply. This process is applied to each operator along the solution path, generating a macro-operator that is guaranteed to lead to the goal state. The method is very similar to that employed by Fikes, Hart, and Nilsson (1972) in their early STRIPS system. An added attraction is that one need not worry about misclassifying side paths that actually lead to the goal by another path; analytic methods do not use negative instances in generating concept descriptions, so this is not a problem. Mitchell et al. (1983) have applied this approach to learning search heuristics for symbolic integration, whereas Minton (1984) has applied it to game-playing domains.

Carbonell has explored a related approach in his work on problem solving by analogy (1983). During its attempt to solve a problem, Carbonell's system retains information not only about the operators it has applied, but about the *reasons* they were applied. Upon coming to a new problem, the system determines if similar reasons hold there, and if so, attempts to solve the current problem by analogy with the previous one. Mitchell's, Minton's,

[8] Neither does any problem arise for bi-directional approaches such as Mitchell's version space method, since these can use the general boundary in proposing moves.

and Carbonell's methods all analyze the solution path in order to take maximum advantage of the available information. In the next section, we will discuss analogical methods in more depth.

4.4 Open Problems in Heuristics Learning

We have seen that heuristics learning can be viewed as the general case of learning from examples, and many of the open problems in this area are closely related to those for concept learning. For instance, one can imagine complex domains for which no perfect rules exist to direct the search process. In such cases, one might still be able to learn probabilistic rules that will lead search down the optimum path in *most* cases. This situation is closely related to the task of learning concepts from noisy data. Similarly, one can imagine attempting to learn search heuristics while extending an incorrect or incomplete representation; Utgoff (1986) and Lenat (1983) have done initial work on this problem.

There are many problem-solving domains in which some moves are better than others, but for which no absolute good or bad moves exist. As with learning from examples, most of the existing heuristics learning systems assume that "all or none" rules exist for operator relevance conditions. Even if one could modify the credit assignment methods to deal with such continuous classifications, it is not clear how one would alter the methods for finding heuristic conditions on operators. Rendell (1983) has explored an alternative approach in which heuristics are represented as numeric evaluation functions and learning consists of discovering those functions, but other approaches need to be explored as well.

5. Learning by Analogy

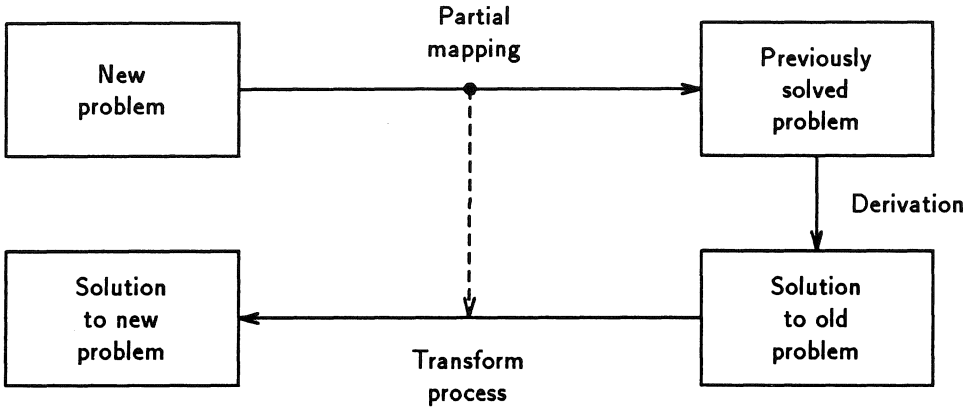
Once a problem has been solved, it can provide useful guidelines and suggestions for solving related problems. In the previous section we saw how search heuristics could be abstracted, but here we focus on using past solutions directly to guide the construction of new solutions. *Analogical problem solving* emulates the human ability to exploit past experience, following the solution of a worked-out example problem to expedite problem solving in new but closely related situations.

5.1 Transformational Analogy

Let us first consider the most direct method of transferring information from past solutions to the new problem: the *transformation analogy* method (Carbonell, 1983). When a new problem is encountered, the method proceeds as follows:

1. *Search* episodic memory of past problem instances for one or more that closely match the current problem description.
2. *Recall* the solution associated with that problem description (or, if more than one, the set of alternative solutions).

3. *Transform* the recalled solution by an incremental process of directed perturbations on the recalled solution, reducing the difference between that which the solution accomplishes and that which the new problem requires. The process can be guided by *means-ends analysis* (Newell & Simon, 1972) in the space of solutions rather than in the space of possible world states (see Carbonell, 1983, for a much more detailed exposition).
4. If the transformation proves impossible, perhaps due to an insurmountable difference between the new and old problems, then select a new candidate analog problem, or abandon the analogy process in favor of direct problem solving methods.



1. Match old problem similar to new one.
2. Recall final solution to the old problem.
3. Transform recalled solution to satisfy the constraints of the new problem.

* Use match to guide transformation.
* Ignore the solution procedure (derivation).

Figure 13. Transformational analogy compares a new problem description against previously solved problems, and transfers the solution of the most similar past problem into a solution to the new problem.

As illustrated in Figure 13, the transformational analogy process exploits past experience, a process of great utility if the types of problems solved earlier are any portent of new problems likely to be encountered in the future. Now let us turn to an example where transformational analogy has proven useful. The problems illustrated in Figure 14 have been borrowed from the work of Anderson and Kline (1979), but here we describe a new experiment with high school sophomores at the very start of their geometry course. Each student was given several problems to solve, the first always being the "RONY" problem in Figure 14: Given that the points R, O, N, and Y are colinear and that RO = NY, prove that RN = OY. Among the set of other problems given to each student (sometimes immediately following "RONY", sometimes towards the end), was the analog problem with

angles: Given that angle BAC is congruent to angle DAE, prove that angles BAD and CAE are also congruent. Analogical transfer required noticing the structural similarity, converting line segments to angles, and confirming that line segment addition could be replaced by the (equally sound) angle addition in the proof structure.

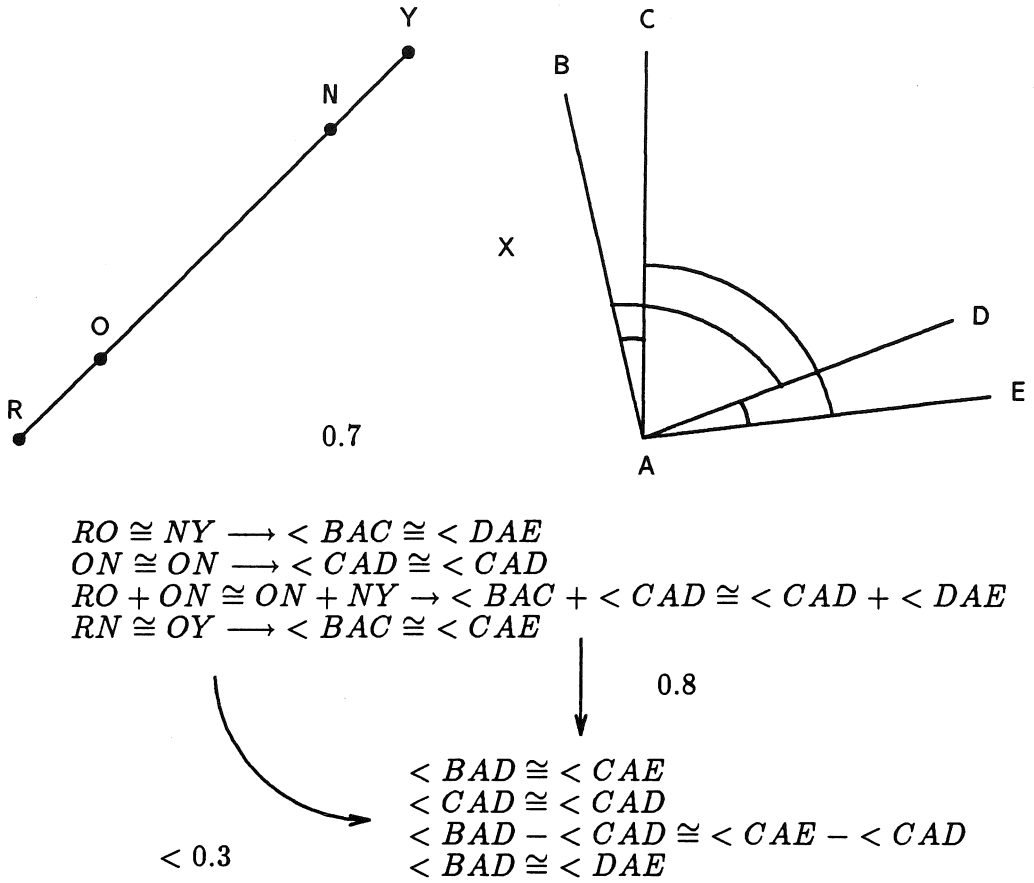


Figure 14. Most students who proved the line segment congruence were able to prove the angle congruence much faster by transformational analogy.

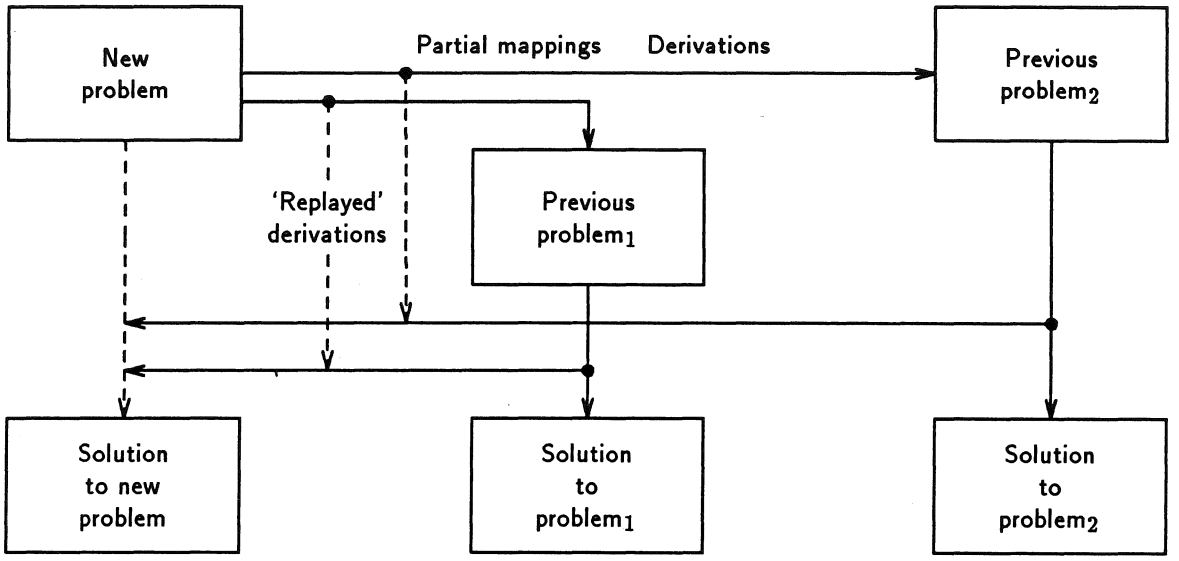
What were the results? Students took 10 to 12 minutes to solve the RONY problem, but the 70% who noticed the analogy required only 2 to 3 minutes to solve the angle analog. The other 30% took once again 10 to 12 minutes. In a later experiment, students were given a variant of the angle problem (after solving the RONY problem) where the large angles BAD and CAE were given as congruent, and were require to prove that the outer angles BAC and DAE were congruent. Here, less than 30% of the students noticed the analogy, but those who did exhibited the same 5-fold speedup in problem solving. However, when students were given both the RONY and the first angle problem to solve prior to the new angle problem, up to 80% established the analogy.

From such experiments one learns that analogy makes human problem solving more effective, but that the problems must be fairly close to each other to have a reasonable expectation that an analogy will be established. Computer implementations of transforma-

tional analogy exhibit similar behavior, though one can trade off the speed of finding an analogy against the breadth of search, enabling the system to find more remote analogies.

5.2 The Derivational Analogy Method

In formulating plans and solving problems, a considerable amount of intermediate information is produced in addition to the resultant plan or specific solution. For instance, formulation of subgoal structures, generation and subsequent rejection of alternatives, and access to various knowledge structures all typically take place in the problem solving process. However, the solution transformation method outlined above ignores such intermediate information, focusing only upon the resultant sequence of instantiated operators corresponding to external actions, and disregarding the reasons for selecting those actions. Carbonell (1986) presents the details of the derivational analogy method, but Figure 15 gives the general idea.



- The derivation is mapped and replayed, not just the solution.
- Multiple past problem-solving episodes can be integrated.
- Recall of previous problems entails partial match or (partial or exact) match of initial segment of derivation.
- Learn strategies, not just generalized plans.

Figure 15. The derivational analogy process: Traces, including decisions and justifications of similar past problem solutions, are replayed and, where necessary, modified to reconstruct a solution to a similar new problem.

The derivational process enables one to draw more distant analogies without violating essential aspects of a problem. This is because the *justifications* for each step in the solution process are preserved, and only if the same justifications hold in the new situation is that step

proposed as part of the derivational transfer. The extra bookkeeping work in maintaining the justification structure – essentially a small truth maintenance system (Doyle, 1979) – is rewarded by the higher quality analogies one is able to draw.

5.3 A Case for Case-Based Reasoning

The vast majority of present-day expert systems encode their knowledge as a large, amorphous set of domain-specific rules (Feigenbaum, Buchanan, & Lederberg, 1971; McDermott, 1980, 1982; Shortliffe, 1986; Waterman, Hayes-Roth, & Lenat, 1983). The "knowledge engineering" task is defined as one of extracting from the human expert the set of rules that comprise his or her expertise in a particular, well-defined domain. The task is by no means easy. It can take years of laborious efforts by teams of domain experts and AI researchers in an iterative process of formulating, evaluating, re-formulating, discarding and refining a set of rules to develop the knowledge base of a particular expert system. Fortunately, the tacit assumption that domain knowledge must necessarily be represented as large sets of context-independent rules is proving to be only an early engineering decision, and a very limiting one at that. The knowledge must be captured, but the question remains as to the best means of representing and acquiring this expertise in a computationally effective manner.

Human experts are incredibly poor at producing general deductive rules that account for their behavior. When forced to do so by insistent knowledge engineers, they try hard and produce faulty rules. When later faced with a problem in which the rule fails, the typical response is: "Well, I didn't think of *that* situation, but perhaps I can fix the rule ...or add a new one ..." This *ad-hoc* iterative process, slow and frustratingly inefficient as it may be, usually converges upon an acceptable knowledge base. However, a much more efficient and humane approach is to let the experts do what they do best: solve problems in their domain of expertise. The only added burden is a reporting requirement. Each problem solving step, including references to static domain knowledge or to heuristics of the domain, must be reported explicitly, along with the reason why such knowledge was used. This process provides external derivational traces that a derivational analogy system can use to solve similar problems in an effective manner. Although the derivational method was originally conceived as a means to reason and learn from the system's own past experience, it works equally well as a means to reason and learn from the experience of a more knowledgeable external source, such as a human expert or a worked-out problem example in a textbook.

Case-based reasoning is particularly prevalent in law – at least in the British and American systems of jurisprudence – and in medical diagnosis and treatment. The idea of case-based reasoning in expert systems is not new. For instance, Schank (1983) advocates this method as superior and closer to human reasoning than present expert systems. Doyle (1984) proposes the notion of emulating the human master-apprentice process as a means whereby the latter (human or computer) can acquire expertise by replicating the reasoning processes of the former. The derivational analogy process is an effective computational mechanism for providing expert systems with the ability to reason from cases, whether the cases be past experience or externally acquired knowledge. Thus, as case knowledge expands, so does the ability to solve more and more problems in the chosen domain of expertise.

5.4 Acquiring Generalized Plans

However, human experts can solve problems progressively more quickly and effectively with repeated experience, suggesting that some knowledge is gradually compiled into more general processes abstracted from the concrete cases. Thus, case-based reasoning may reflect only a crucial intermediate stage in the learning process and may account only for problem solving behavior in infrequently recurring situations. For the most routine recurring problems, the learner moves beyond derivational analogy to produce general plans that can be instantiated directly. This process requires that solutions derived from a common analogical parent form a set of positive exemplars, and unrelated or failed solutions form a set of negative exemplars. These sets are given to a general inductive engine, preferably an incremental one such as Mitchell's (1982) version space method that abstracts a generalized plan from the recurring common aspects of these solutions. Later, the generalized plan can be instantiated directly – or refined further if more instance solutions are derived. Figure 16 summarizes this process, which is discussed at greater length by Carbonell (1983).

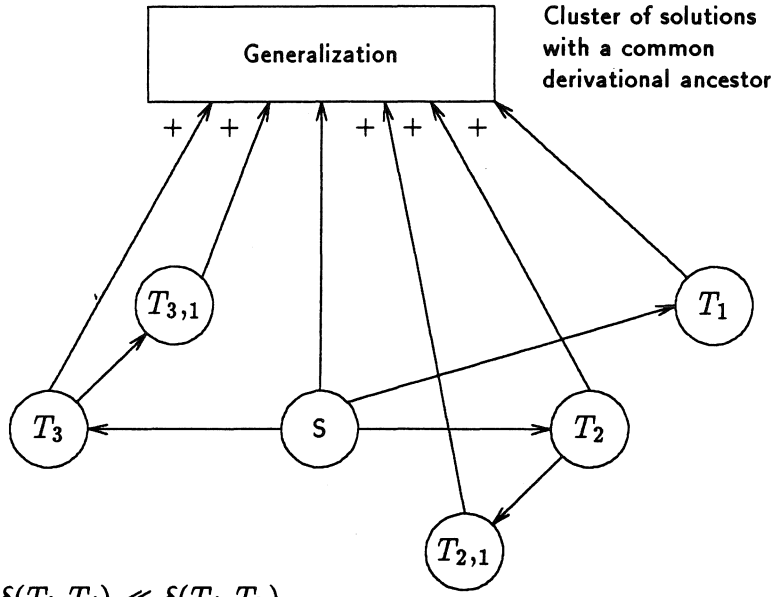


Figure 16. Generalizing plans from analogically related solutions.

For instance, suppose the initial problem was to plan an automobile trip from Boston to Los Angeles, and analogical variants include auto travel between other remote cities in North America. In this case, the generalized plan retains all the common characteristics (such as requiring maps, money, time, a working car, etc.), and abstracts away the varying ones

(such as the source and destination cities, the compass direction of travel, the phase of the moon, etc.). The cluster of derivational (or transformational) children provide the positive exemplars, but where do negative exemplars come from? One source of negative exemplars is members of other clusters; i.e., problems that were effectively solved by different means. A more useful source of *near-miss* negative exemplars are failed analogies; i.e., analogies proposed but not carried through to a solution. For instance, in our automobile travel example, suppose the problem of traveling from Boston to London arose. This matches fairly closely previous members of the cluster, but upon attempting to find a route between the cities, one quickly discovers that the Atlantic Ocean is an insurmountable barrier. Thus, the general plan is constrained to apply only to cities in the same land mass – in much the same way that Winston (1975) exploited near-miss examples to infer crucial discriminant properties in learning about arches.

6. Language Acquisition

A fourth major area of machine learning research has dealt with the acquisition of language. The overall task of language acquisition is very complex and involves many levels, including: learning to recognize and generate words; learning the meanings of words; learning grammatical knowledge; and learning pragmatic knowledge. Each of these subproblems is interesting in its own right, but since the majority of AI work on language acquisition has dealt with grammar learning, we will focus on that issue here. Other reviews of computational approaches to language learning can be found in McMaster, Sampson, and King (1976), Anderson (1977), Pinker (1979), Langley (1982), and Hill (1987).

6.1 Learning Grammars from Sample Sentences

Some of the earliest work in machine learning addressed the problem of grammar acquisition, and this is still an active area of research in the field. The basic task is simply stated: given an initial set of grammatical sentences from some language, find a procedure for recognizing all other grammatical sentences in that language. The induced grammar may take many different forms, including rewrite rules, an augmented transition network, or a production system. Note that one is given only legal sentences from the language to be learned, and that no "negative instances" are provided. Solomonoff (1959), Knowlton (1962), Garvin (1967), and Horning (1969) carried out early research on this problem. Wolff and Berwick have described more recent work in this tradition, and we will focus on their results here.

Wolff (1980) has described SNPR, a program that acquires grammatical knowledge in a very data-driven manner. The system inputs a sequence of letters and generates a phrase-structure grammar (stated as rewrite rules) that summarizes the observed sequence. The program is not provided with any punctuation or pauses between words or sentences; it must determine these boundaries on its own.

The SNPR system carries out a hill-climbing search through the space of possible grammars using two operators – one for forming *disjunctive* classes such as **noun**, and another for defining chunks or *conjunctive* structures, such as **dog**. It also includes operators for

generalization and recursion, but we will not focus on them here. The system employs a numeric evaluation function to determine which of its operators should be applied in a given situation. This function measures two features of the grammar that would result – the *compression capacity* (the degree to which a given grammar compresses the original data) and the *size* of the grammar. At each point in its learning process, SNPR selects that step which gives the greatest improvement in compression capacity per unit increase in size of grammar. Thus, this evaluation function directs the system's search through the space of possible phrase-structure grammars.

One of the interesting aspects of Wolff's system is the manner in which its two operators interact. The system begins by forming chunks for pairs of symbols such as **th** and **ch**. Whenever a chunk is created, the component symbols are replaced by the symbol for that chunk. The process can then be applied recursively to generate hierarchical chunks; this leads to chunks for words such as **the**, **cat**, and **chased**. However, chunks can also be used to form disjunctive classes such as **noun** and **verb**. When this occurs, SNPR substitutes the symbol for this new class for all occurrences of its members; thus, **dog** and **cat** would be replaced with the **noun** symbol. At this point something quite interesting can occur: the system can form chunks in terms of these disjunctive classes, generating terms such as **prepositional-phrase** and **noun-phrase**. Thus, the system begins with a representation involving individual letters and gradually bootstraps itself into a grammar-based representation.

Berwick (1979, 1980) has described LPARSIFAL, a grammar acquisition system that differs substantially from Wolff's SNPR. This system represents its grammatical knowledge as a set of rules, but ones quite different from SNPR's rewrite rules. The program inputs a sequence of legal English sentences, but these sentences differ from Wolff's in that each one consists of separate words. In addition, the sentences themselves are separated from each other. No meanings are associated with either words or sentences.

LPARSIFAL is based on Marcus' (1980) wait-and-see approach to syntactic parsing. In this framework, grammatical expertise is stored as condition-action rules that match against two data structures – an input buffer and stack of partially constructed parse trees. The system employs only a few simple operators, e.g., for creating a node and pushing it onto the stack, moving a node from the stack to the buffer, attaching an item in the buffer onto the stack, and so forth. These operators are applied to an input sentence in sequence until that sentence has been completely parsed.

Berwick's system begins with a knowledge of Chomsky's X-bar theory (1980) and an interpreter for applying grammar rules to parse sentences. When given a new sentence, LPARSIFAL attempts to parse it using its existing rules. If it reaches an impasse, the system attempts to create a new rule that will handle the problem-causing situation. The conditions of the new rule are based on the state of the parse when the problem was encountered, including the top of the stack and the contents of the input buffer. Upon adding the new rule to memory, the system checks to see if any existing rules have identical actions.

If LPARSIFAL finds a rule with the same action,[11] it compares the two condition sides to determine what they hold in common. The resulting mapping is used to construct a more general rule with the same action that replaces the two previous rules. Differing conditions are dropped from the resulting general rule or, in some cases, lead to the creation of syntactic classes like nouns and verbs. Thus, the program's method for combining rules is similar to the specific-to-general method we considered in the context of learning from examples. However, since Berwick employs a simple attribute-value representation, he does not have to worry about search through the space of rules. Because of this simplifying assumption, LPARSIFAL needs no negative instances to eliminate competing hypotheses.

Upon reflection, Berwick's approach to grammar acquisition is reminiscent of another class of learning problems – the task of *heuristics learning*. One can view his system as beginning with a set of operators for parsing sentences, along with legal conditions stated in terms of X-bar theory. However, in order to parse sample sentences, the system must search. When the goal state (an empty input buffer) is achieved, LPARSIFAL assigns credit to each move along the solution path, creating specific heuristics for each situation. Berwick has transformed the grammar learning task into the task of learning search heuristics, a counterintuitive but fruitful approach.[12]

6.2 Learning Grammars from Sentence-Meaning Pairs

Although the grammar-learning task described above has many interesting aspects, it differs from human language acquisition in an important respect. Rather than simply learning grammars for parsing sentences, the human learner acquires grammars for *mapping* sentences onto their meanings. Moreover, we know from child language data that the human learner does not hear sentences in isolation; the sentences usually describe some event or object in the immediate environment. This observation leads to a different formulation of the grammar-learning task: given a set of grammatical sentences from some language, along with the *meaning* for each sentence, find some procedure for mapping sentences onto their meanings or vice versa.

This view of grammar acquisition differs significantly from the first one we examined. Grammatical knowledge must contain more than information about sentence structure – it must also relate this structure to meaning. This alternative view of grammar learning leads to quite different models of the learning process. Kelley (1967), Siklóssy (1972), and Klein and Kuppin (1970) carried out the earliest work in this "semantic" tradition. More recent systems have been described by Hedrick (1976), Reeker (1976), Anderson (1977), Selfridge (1981), Sembugamoorthy (1979), and Langley (1982).

Anderson (1977) developed LAS, a program that learns to understand and generate sentences in both English and French. LAS represents grammatical knowledge as an augmented transition network (ATN), with both semantic and syntactic information stored on each link.

---

[11] This is an oversimplification; in fact, the rule must also have the same X-bar context, but we cannot discuss the details of this context here.

[12] We should note that Berwick reported the first version of LPARSIFAL in 1979, when very few results had been achieved in heuristics learning.

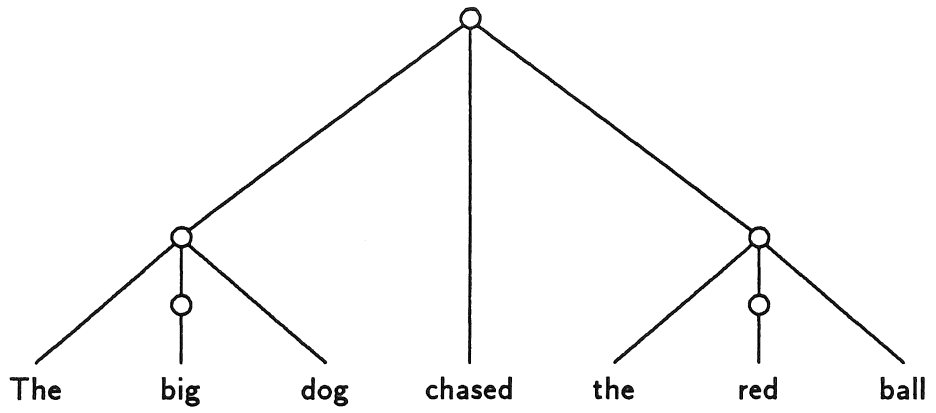


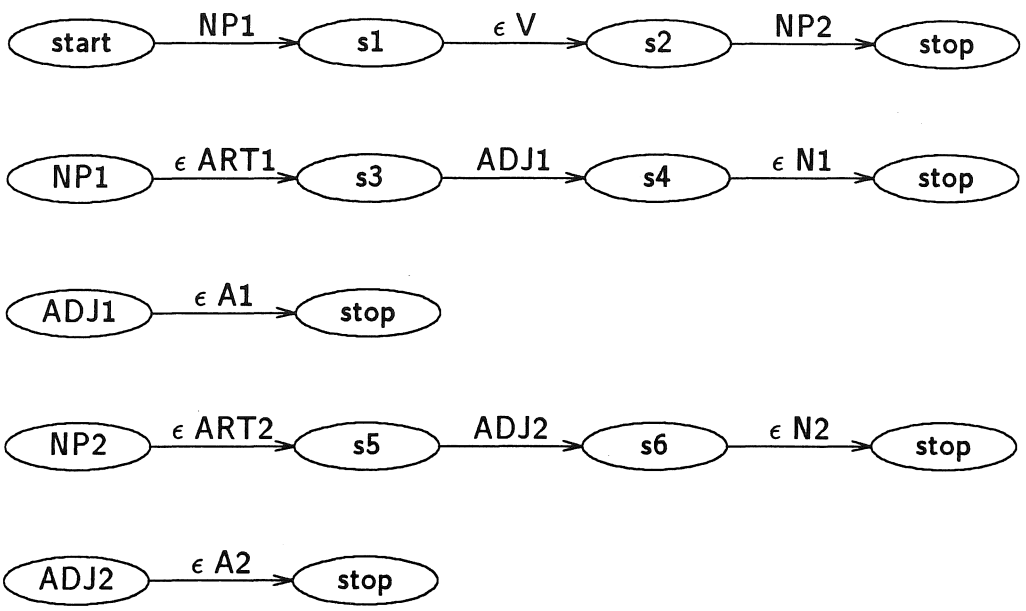
Figure 17. A sample parse tree.

The system accepts legal sentences and their associated meanings as input, with meaning represented in terms of a semantic network. In addition, LAS is provided with the main topic of each sentence, as well as the words associated with various concepts. Finally, the program makes two assumptions about the nature of grammar: (1) that some concepts (like shapes) play the role of nouns; and (2) the *graph deformation condition*, which roughly states that if two words occur near each other in a sentence, the concepts associated with those words must occur near each other in the meaning of that sentence.

These sources of information are sufficient to enable LAS to determine a unique parse tree for any given sentence-meaning pair. For instance, suppose the system is given the sentence **The big dog chased the red ball** and its associated meaning. We can represent this meaning in terms of a semantic network, and we can transform this network into a parse tree like that shown in Figure 17. This can also be represented as the list structure **((The (big) dog) chased (the (red) ball))**, in which parentheses indicate the level of the tree. However, any given semantic network can be translated into a number of such trees, and LAS used its knowledge of the sentence's main topic, the graph deformation condition, and concept-word links to determine a unique tree.

Given the parse tree for a sentence, it is a simple matter to generate a fragment of an augmented transition network that will parse that sentence. For instance, given the parse tree in Figure 17, LAS would transform this structure directly into the (initial) ATN shown in Figure 18. Since the parse tree has three branches at the top level, LAS would generate a top-level ATN with three links – one for the first structure **(The (big) dog)**, one for the second structure **chased**, and one for the third **(the (red) ball)**. Since the first and third components themselves contain internal structure, LAS would build a sub-ATN for both of these, each with three links, and so forth until it reached the terminal nodes.

After it has constructed an initial ATN, LAS attempts to incorporate new parse trees with as little modification as possible. For instance, given the new sentence **The small cat chased the long string**, the system would note that its ATN would parse this quite well, if only certain classes were expanded. In this case, the class **ADJ1 = {big}** must be extended to **ADJ1 = {big, small}**, the class **NOUN1 = {dog}** must be extended to **NOUN1 = {dog, cat}**, and so forth. In addition to expanding word classes, LAS employs two other



V = {chased} N1 = {dog} N2 = {ball}
ART1 = {the} ART2 = {the}
A1 = {big} A2 = {red}

Figure 18. Initial ATN based on a single sentence.

learning mechanisms. First, when the system finds two word classes that share a significant number of elements, it combines them into a single class. Second, if LAS finds two sub-ATNs to be sufficiently similar, it combines them into a single subnetwork. In certain cases this process leads to recursive networks, such as those used in parsing noun phrases. These steps occasionally lead the system to learn overly general ATNs, and LAS has no way to recover from these errors.

Now let us turn to Langley's (1982) AMBER, a cognitive simulation of the early stages of child grammar acquisition. Like LAS, this system accepts sentence/meaning pairs as input, using a semantic network to represent meaning. The program also shares LAS's requirement that the meanings content words (such as **ball** and **bounce**) be known, and that the main topic of each sentence be available. However, AMBER differs from Anderson's system by representing grammatical knowledge as production rules that generate sentences from meaning structures. Although Langley's system does not assume the graph deformation condition, an analogous constraint arises from the system's strategy for generating sentences.

AMBER uses its knowledge of each sentence's main topic to transform its semantic network into a tree structure, and then proceeds to generate an utterance to describe the structure. In doing so, it employs the notion of goals and subgoals. The top level goal is to describe the entire tree, but to achieve this goal, the system creates subgoals to describe nodes lower in the tree. At the outset, AMBER can handle only one subgoal at a time,

leading the system to generate one-word "sentences." Much of the system's learning consists of acquiring rules that let it deal with multiple subgoals, and then identifying the relative order in which those subgoals should be achieved. However, the system never returns to a goal once it has been deactivated; it may thus omit words (as do children), but it will never generate sentences that violate the graph deformation condition.

The AMBER system begins with the ability to say one content word at a time. Based on differences between these utterances and the sample sentences it is given, the model generates new rules that let it generate combinations of content words in the correct order. AMBER also uses a discrimination process to determine the conditions under which it should produce function words like **is** and the suffix **ing**.[13] In both cases, the acquired rules must be relearned a number of times before gaining enough "strength" to take over control from the default rules. Taken together, these mechanisms replicate a number of child language phenomena, including word omissions, the gradual disappearance of such omissions, and the order in which function words are mastered.

6.3 Negative Instances in Grammar Learning

Before moving on, let us consider the role of negative instances in grammar learning. We have seen that many learning methods rely on negative instances to direct their search through the space of hypotheses. For instance, specific-to-general condition-finding methods employ such instances to eliminate overly general hypotheses. Similarly, general-to-specific condition-finding methods use negative instances to determine how overly general descriptions should be made more specific. Negative instances are heavily used in learning from examples, where they are provided by a tutor, and in heuristics learning, where they arise from steps leading off the solution path.

Of the grammar learning systems we have discussed, only Langley's AMBER actually employs negative instances, but Reeker's (1976) PST and Anderson's (1981) ALAS have also used this type of information. This may seem odd, since these models are only given *legal* sentences as examples. However, AMBER and its relatives do not learn rules directly at the sentence level, but focus instead on the *parts* of sentences.[14] Moreover, these systems learn to *map* sentences onto their meanings (and vice versa), and this lets them make predictions which may prove incorrect.

To clarify the point, consider an example in which AMBER predicts that **ing** should occur after the word **bounce**. If this does not occur in the adult utterance, the system can label this situation as a negative instance and use it to determine the appropriate generation condition for the **ing** suffix. Positive instances can be generated in an analogous fashion, based on successful predictions. All this does *not* mean that children receive negative evidence in

[13] This process is closely related to the general-to-specific method for learning from examples that we described in an earlier section.

[14] Berwick's (1979, 1980) LPARSIFAL could also have generated negative instances by noting which actions failed to allow a successful parse. However, the system's search was already sufficiently constrained that it did not use this information.

the form of ungrammatical sentences. However, it *does* mean that one can *generate* negative instances from discrepancies arising between predicted and actual sentences paired with their meanings, and one can use such negative instances to constrain the process of grammar acquisition.

7. Learning by Discovery

Most of the methods we have examined involve some form of an external tutor or internal problem solving traces that provide the information necessary for learning. However, humans encounter many situations in which they must discover regularities in their environment through observation and experimentation. This is the task confronting the scientist trying to discover new facts and formulate new theories; we will rely on this analogy in our discussion of machine discovery. Of course, scientific discovery is a complex process, involving activities ranging from the design of experiments and the construction of measuring instruments to the generation and testing of explanatory theories. Here, we limit our treatment to two of the discovery problems that have received recent attention within the machine learning community – the formation of classificatory taxonomies and the discovery of empirical laws that describe regularities in observed data.

7.1 Taxonomy Formation and Conceptual Clustering

Before the scientist can discover empirical laws and formulate theories, he must first decide upon some classification scheme for the objects under study. For example, chemists made little progress until they could distinguish between different elements, such as gold and lead. Later progress occurred after they partitioned these substances into *classes* such as metals, inert gases, and acids. Similarly, theories of evolution rested upon taxonomies formulated by early biologists like Linnaeus.

Figure 19 illustrates the task of taxonomy formation using the two-bodied cells described earlier. Given the thirteen cells shown in the figure, one must generate some taxonomic hierarchy that groups these cells into classes, subclasses, and so forth. There are many ways to organize these data, but some may be preferred to others. The figure shows one such partitioning, marking the two major classes with solid rectangles and marking the four subclasses with dotted rectangles.

The earliest work on automated taxonomy formation was not carried out by AI researchers, but rather by statisticians and biologists who developed the methods of *cluster analysis* and *numerical taxonomy.* These algorithms accept as input some set of objects and their associated descriptions and generate a hierarchical classification tree that summarizes the data. Typically, such methods use attribute-value representations for objects, viewing them as points in an n-dimensional space. The similarity between two objects or two clusters is measured by their distance in this space, and the methods attempt to find the taxomonic scheme that maximizes intra-cluster similarity and minimizes inter-cluster similarity. Table 6 presents one version of the numerical taxonomy approach. However, many variations exist with different measures of distance, and these often lead to different partitions of the same data.

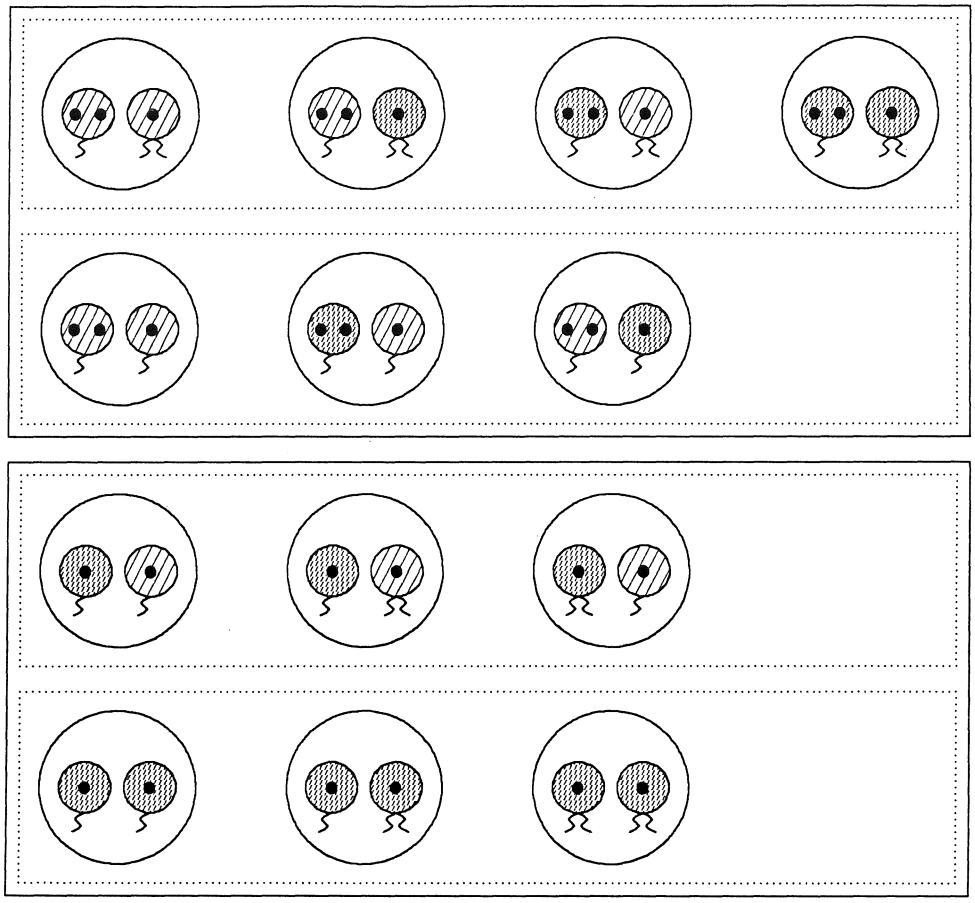


Figure 19. The problem of taxonomy formation.

The work on numerical taxonomy formation is quite interesting from an AI perspective, since it clearly takes a heuristic approach. However, Michalski and Stepp (1983) have argued that numerical taxonomy methods suffer from two limitations. First, these methods generate only extensional definitions of categories, and one would like much more concise intensional definitions with predictive power. Second, the methods use only the objects themselves in evaluating alternative clusters, and one would like to use the intensional definitions of objects as part of the evaluation criterion (e.g., preferring simpler to more complex descriptions).

In response, Michalski and Stepp (1983) have formulated the related task of *conceptual clustering*. In this task, one is still presented with a set of objects and their associated descriptions, and one must still generate a hierarchy containing clusters of objects. However, one must also generate intensional descriptions for those clusters, and competing clusters must be evaluated according to the quality of their associated descriptions. They have argued that the resulting clusters should be more conceptually coherent than those generated by the traditional methods.

Consider the taxonomic hierarchy presented in Figure 20, which summarizes the clusters given in Figure 19. This taxonomy goes beyond the simple extensional definitions we had before; it also provides an intensional description for each concept in terms of the defining

TABLE 6
A Numerical Taxonomy Method

1. Find the two closest objects and create a cluster that contains them as members.
2. Replace the clustered objects with the new cluster, treating it as a new object whose coordinates are the weighted arithmetic average of its members' coordinates.
3. If all objects are covered by a single cluster, then halt; otherwise go to step 1.

features for that class.[15] Thus, we see that one major category covers cells with one nucleus in each body, while the other covers cells with two nuclei in one body and one nucleus in the other. The subclasses of the first category include additional conditions about the body colors, while the subclasses of the second category refer to the number of tails. Note that these concept definitions lead to predictions about other cells that have not yet been observed.

Although the conceptual clustering task bears some similarity to the problem of learning from examples, there are three important differences. First, in conceptual clustering there is no tutor to place objects into classes; the learner must solve this clustering problem on his own. Second, the resulting taxonomy involves disjunctive classes; a conjunctive clustering task would be one in which only a single object was observed, and would not be very interesting. Third, conceptual clustering systems must form concepts at multiple levels of description; in addition to describing a set of concepts, the learner must impose some hierarchical organization on those concepts.

7.2 Methods for Conceptual Clustering

There are a variety of approaches to the conceptual clustering problem, though we will review only one of them in detail here. We have selected Fisher's (1985) RUMMAGE system, since its basic method is easy to communicate. Table 7 provides an English paraphrase of the clustering mechanism. RUMMAGE assumes that objects are described in terms of attribute-value pairs, and it uses this knowledge to form potential clusters. The system constructs its taxonomy in a top-down fashion, at each point selecting one attribute to divide objects into clusters and using a general-to-specific learning from examples method to generate an intensional description for each cluster. The attribute producing the best[16] intensional descriptions is selected, and its values are used to create the initial branches in the taxonomy. Objects are sorted down these branches depending on their values, and the process is applied recursively to generate lower level clusters. This process continues until the quality of the cluster descriptions falls below a user-specified threshold.

[15] This structure does *not* represent a search tree through the space of concept descriptions; it represents the *output* of a conceptual clustering system.

[16] Each candidate clustering is evaluated on two measures: maximal simplicity and minimal overlap, roughly analogous to intra-cluster and inter-cluster measures from numerical taxonomy.

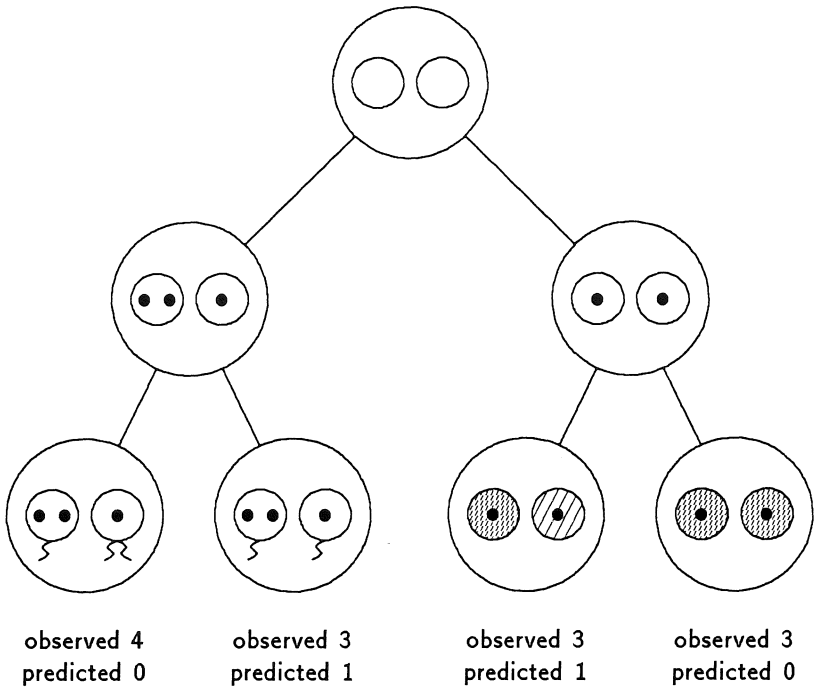


Figure 20. A taxonomic hierarchy.

The RUMMAGE algorithm has many similarities to that used by ID3 to construct decision trees from examples. The main difference lies in the evaluation function used by the two systems. ID3's function requires instances to be grouped into positive and negative classes, whereas RUMMAGE generates descriptions of each class and evaluates these instead. However, both systems construct trees in a top-down fashion and both avoid significant search by selecting the "best" attribute at each level. Neither is guaranteed to find the optimal tree, but both are efficient compared to other, more complex learning systems.

RUMMAGE differs significantly from Michalski and Stepp's (1983) CLUSTER/2, one of the earliest conceptual clustering systems. Fisher's program uses its knowledge of attributes and their values to generate potential clusterings. This model-driven approach is efficient, but limits RUMMAGE to forming *monothetic* classification schemes in which only one attribute is used to index each category. Thus, the system could not produce the taxonomy in Figure 20, since two features are introduced at each level. In contrast, CLUSTER/2 uses an iterative method (similar to hill climbing) to search the space of possible clusterings, using concept descriptions (generated by a learning from examples technique) to direct the search process. This approach is much more expensive, but Michalski and Stepp's system can generate *polythetic* hierarchies, in which conjunctions of features define each class.

As with other learning tasks, methods for conceptual clustering can vary along a number of dimensions. For instance, although both RUMMAGE and CLUSTER/2 construct taxonomies in a top-down fashion, one can imagine systems that create conceptual hierarchies from the bottom up. In fact, the numerical taxonomy method we examined earlier operates in exactly this fashion. Similarly, methods can vary in their approach to forming clusters; we have already seen that RUMMAGE uses a model-driven method based on knowl-

edge of attributes, while CLUSTER/2 uses an iterative, successive approximation approach. In contrast, our numerical taxonomy method used a 'greedy' algorithm with inter-cluster distances as its evaluation function. The space of clustering methods is a large one, and we refer the reader to Fisher and Langley (1986) for a fuller treatment of the possibilities.

TABLE 7
RUMMAGE's approach to conceptual clustering

1. Create the top node of the taxonomic hierarchy.
2. For each attribute, sort objects according to the values of that attribute.
3. For each value of an attribute, generate a concept description for objects with that value.
4. Select that attribute with the "best" descriptions (the simplest and least similar).
5. Create branches from the current node for each value of this attribute, and sort objects down these branches to the new nodes.
6. Apply the method to the resulting subsets of objects, recursively selecting attributes and constructing subtrees until their quality falls below threshold.

Although most work on conceptual clustering has assumed that all data are present at the outset, one can also imagine systems that operate in an incremental fashion. In fact, Lebowitz (1983) has described UNIMEM, an incremental system for generating conceptual hierarchies. This system also constructs trees in a top-down fashion, but, in addition, it retains the ability to reorganize its taxonomy as it observes new objects. Fisher (1987) has described COBWEB, another incremental system that uses a probabilistic representation for concepts. As we have seen, most learning systems assume that concepts must be described by a set of necessary and sufficient conditions, but COBWEB instead stores the probability that a given feature will occur for an instance of a concept. The system uses these probabilities to direct its search through the space of conceptual hierarchies.

We believe that incremental, probabilistic methods for concept learning have advantages over traditional approaches, and that they will draw more attention in the future. In addition, most existing methods are limited to attribute-value representations, and we need to explore extensions that will handle more complex relational descriptions. Stepp and Michalski (1986) have described an extension to CLUSTER/2 that addresses this issue. Another direction for future research lies in using functional knowledge to direct the search for taxonomies. Nelson (1973) has argued that children's very early concepts are often functional in nature. For example, a ball is something that one can bounce, and a chair is something that one can sit on. Only later, Nelson claims, are structural features added to these concepts. This suggests that a child's *goals* play an important role in the way he organizes his view of the world. This suggests a potential connection to explanation-based methods for learning from examples, which transform functional concept definitions into structural ones. One might apply these methods to the conceptual clustering task, yielding systems with a quite different flavor than have been explored to date.

7.3 Discovering Qualitative Laws

One important form of empirical law involves qualitative relations between objects. For instance, early chemists found that certain classes of substances (such as acids and alkalis) reacted with each other, while other substances did not. These qualitative relations preceded the discovery of quantitative regularities, forming the framework within which the latter were stated. Relatively little work has been done on qualitative discovery within the folds of machine learning, but we will consider two systems here – Lenat's AM (1977, 1983) and Langley, Zytkow, Simon, and Bradshaw's GLAUBER (1987).

AM operates in the domain of number theory, starting with some 100 initial concepts such as set membership, cardinality, set union, and so forth. The system is also provided with several hundred heuristics for proposing new concepts and conjectures, for gathering data, and for deciding which concepts are "interesting." For example, one heuristic marks concepts with only a few examples (but more than a singleton set) as interesting. If examples of a concept are too hard to find, AM proposes more general versions of that concept; if they are too easy to find, it proposes more specific versions. Similarly, equivalent concepts that are discovered by different paths are marked as interesting, and thus are given preference as the building blocks for yet newer concepts. Lenat's system carries out an agenda-driven best-first search through the space of mathematical concepts and conjectures, directing this search with its measure of interestingness.

When AM was provided with the basic objects and operations of number theory, it rediscovered a number of familiar concepts, including integers, addition, multiplication, factors, and prime numbers. In addition, it conjectured that any integer can be expressed as a unique product of primes (the Unique Factorization Theorem) and that any even integer can be represented as a sum of two primes (Goldbach's conjecture). These conjectures are qualitative laws that relate concepts generated by the system.

Unlike Lenat's AM, the GLAUBER system begins with very little knowledge of its domain. This program inputs a set of facts, such as the tastes of chemical substances and the reactions in which they take part. From these data, the system generates classes of substances (such as acids, alkalis, and salts) and qualitative laws that relate these classes to each other. These laws may contain universal or existential quantifiers and may be combined to express more complex qualitative laws.

GLAUBER carries out a greedy search through the space of classes, basing its decisions on commonly recurring relations. For instance, if the substance HCl reacts both with NaOH and with KOH, the system would consider defining a new class of substances (say alkalis) with NaOH and KOH as members. Upon doing so, it would also formulate one or more qualitative laws based on facts that contain those substances (such as "HCl reacts with alkalis"). This process is applied recursively to form other classes (say acids) and more abstract laws (such as "acids react with alkalis"). The resulting laws make predictions, and if enough of these predictions are observed, the program includes a universal quantifier on the classes (e.g., "for all acids and for all alkalis, acids react with alkalis").

Although they differ along many dimensions, both AM and GLAUBER carry out search through a space of concepts and qualitative laws, and both use heuristics to direct that

search. In some sense, both systems are also forming taxonomies, since they cluster objects and generate laws that "define" those classes. However, they differ from conceptual clustering systems in that these "definitions" describe relations between classes, rather than describing each class in isolation. In this sense, they move beyond simple clustering into the realm of qualitative discovery.

7.4 Discovering Quantitative Laws

Another important aspect of discovery involves the postulation of quantitative laws that summarize numeric data. Again, relatively little work has been done in automating quantitative discovery processes in AI, but BACON.4 (1983) is perhaps the best known example. Given the values of symbolic and numeric variables (e.g., the pressure, volume, and temperature of a gas), the system formulates empirical laws that relate the numeric variables (e.g., PV/T = 8.32). BACON.4 has rediscovered numerous laws from the history of physics and chemistry, including the ideal gas law, Coulomb's law, Snell's law of refraction, Black's heat law, the law of constant proportions, and conservation of momentum. In discovering these laws, the system also postulates a number of intrinsic properties, such as mass, index of refraction, specific heat, and atomic weight.

BACON's discovery method consists of a number of interacting techniques. The system begins by gathering data in a systematic fashion, varying one independent term at a time and examining the values of dependent variables. After gathering a set of values, BACON looks for monotonic relations between terms, uses these to define new terms, and recurses until it finds terms with constant values. After finding laws that hold in a given context, the system varies another independent term, using the constants found at the previous level as dependent terms at this higher level of description. This process continues until all terms have been incorporated into some law.

In cases where BACON encounters nominal (symbolic) independent terms, it postulates intrinsic properties based on the values of some dependent term and looks for a law involving the new property. The first law found in this manner is tautological, but the same intrinsic values are carried over to other situations, leading to empirically meaningful relations. Each intrinsic property has an associated set of conditions under which its values are retrieved. In cases where generalizing these retrieval conditions is not justified by the data, the system may still note common divisors among the inferred intrinsic values. This method proves quite useful in chemistry, where common divisors historically suggested a number of concepts, including atomic weight.

BACON.4's method for finding constant terms is sufficiently simple that we can describe it by three straightforward heuristics:

1. If term X has near-constant values, then formulate a law involving X;
2. Else, if X increases as Y increases, consider the ratio X/Y and go to 1;
3. Else, if X increases as Y decreases, consider the product XY and go to 1.

Table 8 presents a simple example of BACON's application of this method in discovering Kepler's third law of planetary motion. This law can be stated as $D^3 = k\,P^2$, where D

is the distance of a body from its primary and P is the period of that body. The table presents Borelli's original data for Jupiter's satellites, which contain a substantial amount of variation. BACON.4 begins by noting that D and P increase together, leading it to consider the ratio D/P. This term is not constant, but its values decrease as those of D increase; this leads BACON to define the product D^2/P. Again, the values of this variable are not constant, but its values increase as those of D/P decrease. As a result, the program considers the term D^3/P^2. The values of this attribute are constant (within the acceptable range of 7.5 per cent), so BACON formulates a law to this effect. The same method can be used to discover a variety of numeric laws.

TABLE 8
Discovering Kepler's Third Law of Planetary Motion

moon	distance D	period P	D/P	D^2/P	D^3/P^2
A	5.67	1.769	3.203	18.153	58.15
B	8.67	3.571	2.427	21.036	51.06
C	14.00	7.155	1.957	27.395	53.61
D	24.67	16.689	1.478	36.459	53.89

Two more recent empirical discovery systems move beyond BACON's abilities by formulating conditional laws that hold in different situations. Falkenhainer and Michalski's (1986) ABACUS combines BACON-like methods for numerical discovery with condition-finding methods like those used in Michalski's (1980) AQ11 system. ABACUS first finds laws that cover some subset of the data, and then searches for a symbolic description that describes the conditions under which the law holds. The program repeats this process until it has covered as much of the data as possible. ABACUS' authors have also explored new methods for effectively searching the space of numeric laws. Unlike BACON, this system can find complex laws solely on the basis of observational data.

Zytkow's FAHRENHEIT (1987) employs a different approach to discovering conditional laws. Once it has found a numeric relation, the system systematically gathers data until it identifies the range over which the law holds. The upper and lower limits become new dependent terms at the next higher level of description, and FAHRENHEIT looks for laws involving these limit points, just as it does for other variables. Once it discovers such a limit law, the program identifies limits in its range as well, recursively applying the method to ever higher levels of description.

7.5 Description and Explanation

Discovery is a complex phenomena, and it certainly involves more than the mechanisms of taxonomy formation and empirical discovery we have discussed. In particular, it includes the process of constructing *explanations* that account for empirical laws. These include both structural explanations, such as the atomic theory, and process explanations, such as the

kinetic theory of gases. Because of the complexity of the problem, few machine learning researchers have addressed these issues, though some work has been done on structural models (Langley, Zytkow, Simon, & Bradshaw, 1986). Of course, there are different levels of explanation, and one can argue that even numeric laws of the type found by BACON have some explanatory aspects. However, our intuitions tell us that more is involved.

Recent research suggests some directions in which to look for a theory of explanatory discovery. One of these is the area of explanation-based learning, which we described in an earlier section. This work provides a clean definition of what we mean by "explanation," and this definition may prove useful in modeling explanatory discovery. However, explanation-based learning involves the transformation of a functional definition into a structural one. In science, one must perform the inverse mapping, that is, *infer* the explanation from its observed external manifestations. Thus, one must decide that gases are similar to billiard balls, and that the heat of a gas alters the velocity of those balls.

Before we can construct systems that will infer such processes, we must be able to represent the process models themselves. Fortunately, recent work on qualitative physical models (Forbus, 1983; DeKleer & Brown, 1983) provides a framework for such an effort. Moreover, research on reasoning by analogy (Carbonell, 1983; Clement, 1982) might be extended into the development of methods for mapping macroscopic physical models (bouncing billiard balls) onto microscopic explanations (the kinetic theory of gases). Designing and implementing AI systems for explanatory discovery will not be easy, but many of the building blocks are present and this seems a promising direction for future research.

8. Concluding Remarks

In this paper, we examined a range of techniques studied by researchers in machine learning – learning from examples, learning search strategies, language acquisition, and machine discovery – that lay the foundation for symbolic approaches to machine learning. A number of common themes emerged from this examination. We found that much of learning can be viewed as search through a space of concept descriptions, and we considered various methods for directing search through this space. We also found that learning from examples can be viewed as a simpler version of the more complex tasks of learning search heuristics and conceptual clustering, in that credit assignment is simplified and direct feedback is present. For each method that we examined, we found open issues that remain to be explored, including the need for employing functional or causal information to direct the learning process.

Despite its recent emergence, machine learning has developed a variety of well-defined problems that promise to keep researchers occupied for years to come. One major goal for future research involves developing integrated architectures for problem solving and learning that can address many different learning tasks. Anderson's (1983) ACT system falls into this class of architectures, and the SOAR theory of Laird, Rosenbloom, and Newell (1986) is another recent example. However, we need to explore the space of such architectures, just as earlier researchers have explored the space of methods for learning from examples and heuristics learning. In this way, we may ultimately come to understand the nature of learning and the role it plays in intelligent behavior.

References

Anderson, J. R. (1977). Induction of augmented transition networks. *Cognitive Science*, *1*, 125–157.

Anderson, J. R., & Kline, P. J. (1979). A learning system and its psychological implications. In *Proceedings of the Sixth International Joint Conference on Artificial Intelligence* (pp. 16–21). Tokyo, Japan: Morgan Kaufmann.

Anderson, J. R. (1981). A theory of language acquisition based on general learning principles. In *Proceedings of the Seventh International Joint Conference on Artificial Intelligence* (pp. 165–170). Vancouver, BC, Canada: Morgan Kaufmann.

Anderson, J. R. (1983). *The architecture of cognition.* Cambridge, MA: Harvard University Press.

Anzai, Y. (1978). Learning strategies by computer. In *Proceedings of the Second Biennial Conference of the Canadian Society for Computational Studies of Intelligence* (pp. 181–190). Toronto, Ontario, Canada.

Berwick, R. (1979). Learning structural descriptions of grammar rules from examples. In *Proceedings of the Sixth International Conference on Artificial Intelligence* (pp. 56–58). Tokyo, Japan: Morgan Kaufmann.

Berwick, R. (1980). Computational analogues of constraints on grammars: A model of syntactic acquisition. In *Proceedings of the 18th Annual Conference of the Association for Computational Linguistics* (pp. 49–53). Toronto, Ontario, Canada.

Brazdil, P. (1978). Experimental learning model. In *Proceedings of the Third AISB/GI Conference* (pp. 46–50). Hamburg, West Germany.

Carbonell, J. G., Michalski, R. S., & Mitchell, T. M. (1983). Machine learning: A historical and methodological analysis. *AI Magazine*, *4*, 69–79.

Carbonell, J. G. (1983). Learning by analogy: Formulating and generalizing plans from past experience. In R. S. Michalski, J. G. Carbonell, & T. M. Mitchell (Eds.), *Machine learning: An artificial intelligence approach.* Los Altos, CA: Morgan Kaufmann.

Carbonell, J. G. (1986). Derivational analogy: A theory of reconstructive problem solving and expertise acquisition. In R. S. Michalski, J. G. Carbonell, & T. M. Mitchell (Eds.), *Machine learning: An artificial intelligence approach* (Vol. 2). Los Altos, CA: Morgan Kaufmann.

Chomsky, N. (1980). *Rules and representations.* New York, NY: Columbia University Press.

Clement, J. (1982). Analogical reasoning patterns in expert problem solving. In *Proceedings of the Fourth Annual Conference of the Cognitive Science Society* (pp. 79–81). Ann Arbor, MI.

De Kleer, J., & Brown, J. S. (1983). Assumptions and ambiguities in mechanistic mental models. In D. Gentner and A. L. Stevens (Eds.), *Mental models.* Hillsdale, NJ: Lawrence Erlbaum.

Dietterich, T. G., & Michalski, R. S. (1983). A comparative review of selected methods for learning from examples. In R. S. Michalski, J. G. Carbonell, & T.M. Mitchell (Eds.), *Machine learning: An artificial intelligence approach.* Los Altos, CA: Morgan Kaufmann.

Doyle, J. (1979). A truth maintenance system. *Artificial Intelligence*, *12*, 231–272.

Doyle, J. (1984). Expert systems without computers. *AI Magazine*, *5*, 59–63.

Falkenhainer, B. C. (1985). Proportionality graphs, units analysis, and domain constraints: Improving the power and efficiency of the scientific discovery process. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence* (pp. 552–554). Los Angeles, CA: Morgan Kaufmann.

Falkenhainer, B. C., & Michalski, R. S. (1986). Integrating quantitative and qualitative discovery: The ABACUS system. *Machine Learning*, *1*, 367–401.

Feigenbaum, E. A., Buchanan, B. G., & Lederberg, J. (1971). On generality and problem solving: A case study using the DENDRAL program. In *Machine Intelligence* (Vol. 6). Edinburgh: Edinburgh University Press.

Fisher, D. (1987). *Knowledge acquisition via incremental conceptual clustering.* Doctoral dissertation, Department of Information and Computer Science, University of California, Irvine.

Fisher, D., & Langley, P. (1985). Approaches to conceptual clustering. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence* (pp. 691–697). Los Angeles, CA: Morgan Kaufmann.

Fisher, D., & Langley, P. (1986). Conceptual clustering and its relation to numerical taxonomy. In W. A. Gale (Ed.), *Artificial intelligence and statistics.* Reading, MA: Addison-Wesley.

Forbus, K. (1983). Qualitative reasoning about space and time. In D. Gentner & A. L. Stevens (Eds.), *Mental models.* Hillsdale, NJ: Lawrence Erlbaum.

Garvin, P. I. (1967). The automation of discovery procedure in linguistics. *Language*, *43*, 172–178.

Hedrick, C. (1976). Learning production systems from examples. *Artificial Intelligence*, *7*, 21–49.

Hill, J. C. (1987). Language acquisition. In S. Shapiro (Ed.), *Encyclopedia of artificial intelligence* (Vol. 1). New York: John Wiley & Sons.

Hinton, G. E., Sejnowski, T. J., & Ackley, D. H. (1984). *Boltzmann machines: Constraint satisfaction networks that learn* (Technical Report CMU-CS-84-119). Pittsburgh, PA: Carnegie-Mellon University, Department of Computer Science.

Holland, J. H. (1975). *Adaptation in natural and artificial systems.* Ann Arbor, MI: University of Michigan Press.

Holland, J. H. (1986). Escaping brittleness: The possibilities of general-purpose learning algorithms applied to parallel rule-based systems. In R. S. Michalski, J. G. Carbonell, & T. M. Mitchell (Eds.), *Machine Learning: An artificial intelligence approach* (Vol. 2). Los Altos, CA: Morgan Kaufmann.

Horning, J. J. (1969). *A study of grammatical inference* (Technical Report No. CS 139). Stanford, CA: Stanford University, Department of Computer Science.

Hunt, E. B., Marin, J., & Stone, P. J. (1966). *Experiments in induction.* New York: Academic Press.

Jones, R. (1986). Generating predictions to aid the scientific discovery process. In *Proceedings of the Fifth National Conference on Artificial Intelligence* (pp. 513–517). Philadelphia, PA: Morgan Kaufmann.

Kahn, G. S. (1986). Knowledge acquisition: Investigations and general principles. In T. M. Mitchell, J. G. Carbonell, & R. S. Michalski (Eds.), *Machine learning: A guide to current research.* Hingham, MA: Kluwer Academic Press.

Kelley, K. L. (1967). *Early syntactic acquisition* (Technical Report P-3719). Santa Monica, CA: The Rand Corporation.

Kibler, D. F., & Porter, B. W. (1983). Perturbation: A means for guiding generalization. In *Proceedings of the Eighth International Joint Conference on Artificial Intelligence* (pp. 415–418). Karlsruhe, West Germany: Morgan Kaufmann.

Klein, S., & Kuppin, M. A. (1970). *An interactive, heuristic program for learning transformational grammars* (Technical Report No. 97). Madison: University of Wisconsin, Department of Computer Science.

Knowlton, K. (1962). *Sentence parsing with a self–organizing heuristic program.* Doctoral dissertation, Massachusetts Institute of Technology, Cambridge, MA.

Laird, J. E., Rosenbloom, P. S., & Newell, A. (1986). Chunking in SOAR: The anatomy of a general learning mechanism. *Machine Learning*, *1*, 11–46.

Langley, P. (1982). Language acquisition through error recovery. *Cognition and Brain Theory*, *5*, 211–255.

Langley, P. (1983). Learning effective search heuristics. In *Proceedings of the Eighth International Joint Conference on Artificial Intelligence* (pp. 419–421). Karlsruhe, West Germany: Morgan Kaufmann.

Langley, P., Bradshaw, G. L., & Simon, H. A. (1983). Rediscovering chemistry with the BACON system. In R. S. Michalski, J. G. Carbonell, & T. M. Mitchell (Eds.), *Machine learning: An artificial intelligence approach.* Los Altos, CA: Morgan Kaufmann.

Langley, P., Zytkow, J. M., Simon, H. A., & Bradshaw, G. L. (1986). The search for regularity: Four aspects of scientific discovery. In R. S. Michalski, J. G. Carbonell, & T. M. Mitchell (Eds.), *Machine learning: An artificial intelligence approach* (Vol. 2). Los Altos, CA: Morgan Kaufmann.

Langley, P. (1985). Learning to search: From weak methods to domain-specific heuristics. *Cognitive Science*, *9*, 217–260.

Langley, P., Simon, H. A., Bradshaw, G. L., & Zytkow, J. M. (1987). *Scientific discovery: Computational explorations of the creative processes.* Cambridge, MA: MIT Press.

Lebowitz, M. (1983). Concept learning in a rich input domain. In *Proceedings of the International Machine Learning Workshop* (pp. 177–182). Allerton Park, IL.

Lenat, D. B. (1977). Automated theory formation in mathematics. In *Proceedings of the Fifth International Joint Conference on Artificial Intelligence* (pp. 833–842). Cambridge, MA: Morgan Kaufmann.

Lenat, D. B. (1983). The role of heuristics in learning by discovery: Three case studies. In R. S. Michalski, J. G. Carbonell, & T. M. Mitchell (Eds.), *Machine learning: An artificial intelligence approach.* Los Altos, CA: Morgan Kaufmann.

Marcus, M. (1980). *A theory of syntactic recognition for natural language.* Cambridge, MA: MIT Press.

McDermott, J. (1982). R1: A rule–based configurer of computer systems. *Artificial Intelligence*, *19*, 39–88.

McDermott, J. (1982). XSEL: A computer salesperson's assistant. In J. Hayes, D. Michie, & Y-H. Pao (Eds.), *Machine Intelligence* (Vol. 10). Chichester, UK: Ellis Horwood Ltd.

McMaster, I., Sampson, J. R., & King, J. E. (1976). Computer acquisition of natural language: A review and prospectus. *International Journal of Man–Machine Studies*, *8*, 367–396.

Michalski, R. S. (1975). Synthesis of optimal and quasi-optimal variable-valued logic formulas. In *Proceedings of the 1975 International Symposium on Multiple-Valued Logic* (pp. 76–87). Bloomington, IN.

Michalski, R. S., & Larson, J. B. (1978) Selection of most representative training examples and incremental generation of VL_1 hypotheses: The underlying methodology and description of programs ESEL and AQ11 (Technical Report 867). Urbana-Champaign: University of Illinois, Computer Science Department.

Michalski, R. S. (1980). Pattern recognition as rule–guided inductive inference. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *2*, 349–361.

Michalski, R. S., & Stepp, R. (1983). Learning from observation: Conceptual clustering. In R. S. Michalski, J. G. Carbonell, & T. M. Mitchell (Eds.), *Machine learning: An artificial intelligence approach.* Los Altos, CA: Morgan Kaufmann.

Michalski, R. S. (1983). A theory and methodology of learning from examples. In R. S. Michalski, J. G. Carbonell, & T. M. Mitchell (Eds.), *Machine learning: An artificial intelligence approach.* Los Altos, CA: Morgan Kaufmann.

Michalski, R. S., Carbonell, J. G., & Mitchell, T. M. (Eds.) (1983). *Machine learning: An artificial intelligence approach.* Los Altos, CA: Morgan Kaufmann.

Michalski, R. S., Carbonell, J. G., & Mitchell, T. M. (Eds.) (1986). *Machine learning: An artificial intelligence approach* (Vol. 2). Los Altos, CA: Morgan Kaufmann.

Minton, S. N. (1984). Constraint-based generalization. In *Proceedings of the Fourth National Conference on Artificial Intelligence* (pp. 251–254). Austin, TX: Morgan Kaufmann.

Minton, S. N., Carbonell, J. G., Etzioni, O., Knoblock, C. A., & Kuokka, D. R. (1987). Acquiring effective search control rules. In *Proceedings of the Fourth International Workshop on Machine Learning* (pp. 122–133). Irvine, CA: Morgan Kaufmann.

Mitchell, T. M. (1978). *Version spaces: An approach to concept learning.* Doctoral dissertation, Department of Electrical Engineering, Stanford University, Stanford, CA.

Mitchell, T. M. (1982). Generalization as search. *Artificial Intelligence*, *18*, 203–226.

Mitchell, T. M., Keller, R. M., & Kedar–Cabelli, S. (1986). Explanation-based generalization: A unifying view. *Machine Learning*, *1*, 47–80.

Mitchell, T. M., Utgoff, P., & Banerji, R. B. (1983). Learning problem solving heuristics by experimentation. In R. S. Michalski, J. G. Carbonell, & T. M. Mitchell (Eds.), *Machine learning: An artificial intelligence approach.* Los Altos, CA: Morgan Kaufmann.

Mostow, D. J. (1983). Transforming declarative advice into effective procedures: A heuristic search example. In R. S. Michalski, J. G. Carbonell, & T. M. Mitchell (Eds.), *Machine learning: An artificial intelligence approach.* Los Altos, CA: Morgan Kaufmann.

Nelson, K. (1973). Some evidence for the cognitive primacy of categorization and its functional basis. *Merrill-Palmer Quarterly of Behavior and Development*, *19*, 21–39.

Neves, D. M. (1978). A computer program that learns algebraic procedures by examining examples and working problems in a textbook. In *Proceedings of the Second Biennial Conference of the Canadian Society for Computational Studies of Intelligence* (pp. 191–195). Toronto, Ontario, Canada.

Newell, A., & Simon, H. A. (1972). *Human problem solving.* Englewood Cliffs, NJ: Prentice–Hall.

Ohlsson, S. (1983). A constrained mechanism for procedural learning. In *Proceedings of the Eighth International Joint Conference on Artificial Intelligence* (pp. 426–428). Karlsruhe, West Germany: Morgan Kaufmann.

Pinker, S. (1979). Formal models of language learning. *Cognition*, *7*, 217–283.

Quinlan, J. R. (1982). Learning efficient classification procedures and their application to chess endgames. In R. S. Michalski, J. G. Carbonell, & T. M. Mitchell (Eds.), *Machine learning: An artificial intelligence approach.* Los Altos, CA: Morgan Kaufmann.

Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning*, *1*, 81-106.

Reeker, L. H. (1976). The computational study of language acquisition. In M. Yovits & M. Rubinoff (Eds.), *Advances in computers* (Vol. 15). New York: Academic Press.

Reinke, R. E., & Michalski, R. S. (1986). Incremental learning of decision rules: A method and experimental results. In J. E. Hayes, D. Michie, and J. Richards (Eds.), *Machine Intelligence* (Vol. 11). Oxford: Oxford University Press.

Rendell, L. A. (1983). A doubly layered genetic penetrance learning system. In *Proceedings of the Third National Conference on Artificial Intelligence* (pp. 343–347). Washington, D.C.: Morgan Kaufmann.

Samuel, A. L. (1963). Some studies in machine learning using the game of checkers. In E. A. Feigenbaum & J. Feldman (Eds.), *Computers and thought.* New York: McGraw-Hill.

Schank, R. C. (1983). The current state of AI: One man's opinion. *AI Magazine*, *4*, 1–8.

Selfridge, M. (1981). A computer model of child language acquisition. In *Proceedings of the Seventh International Joint Conference on Artificial Intelligence* (pp. 92–96). Vancouver, BC, Canada: Morgan Kaufmann.

Sembugamoorthy, V. (1979). A paradigmatic language acquisition system: An overview. In *Proceedings of the Sixth International Joint Conference on Artificial Intelligence* (pp. 106–108). Tokyo, Japan: Morgan Kaufmann.

Shortliffe, E. H. (1976). *MYCIN: Computer–based medical consultations.* New York: Elsevier.

Siklóssy, L. (1972). Natural language learning by computer. In H. A. Simon & L. Siklóssy (Eds.), *Representation and meaning: Experiments with information processing systems.* Englewood Cliffs, NJ: Prentice–Hall.

Solomonoff, R. (1959). A new method for discovering the grammars of phrase structure languages. In *Proceedings of the International Conference on Information Processing.*

Stepp, R. E., & Michalski, R. S. (1986). Conceptual clustering: Inventing goal-oriented classifications of structured objects. In R. S. Michalski, J. G. Carbonell, & T. M. Mitchell (Eds.), *Machine learning: An artificial intelligence approach* (Vol. 2). Los Altos, CA: Morgan Kaufmann.

Utgoff, P. E. (1983). Adjusting bias in concept learning. In *Proceedings of the Second International Machine Learning Workshop* (pp. 105–109). Allerton Park, IL

Utgoff, P. E. (1986). *Machine learning of inductive bias.* Hingham, MA: Kluwer Academic Publishers.

Waterman, D., Hayes-Roth, F., & Lenat, D. (Eds.) (1983). *Building expert systems.* Reading, MA: Addison–Wesley.

Winston, P. H. (1975). Learning structural descriptions from examples. In P. H. Winston (Ed.), *The psychology of computer vision.* New York: McGraw-Hill.

Zytkow, J. M. (1987). Combining many searches in the FAHRENHEIT discovery system. In *Proceedings of the Fourth International Workshop on Machine Learning* (pp. 281–287). Irvine, CA: Morgan Kaufmann.